

AutoCAD: программирование и адаптация

ДИАЛОГ·МИФИ

Ю. А. КРЕЧКО

АВТОКАД:
ПРОГРАММИРОВАНИЕ И АДАПТАЦИЯ



"ДИАЛОГ-МИФИ"
МОСКВА-1996

УДК 519.682
К80

Ю. А. Кречко
К80 AutoCAD: программирование и адаптация. — М.: "ДИАЛОГ-МИФИ", 1996. — 240 с.
ISBN 5-86404-060-6

Описан язык программирования Автолисп применительно к Автокаду версии 12, даются способы модификации меню, приемы разработки диалоговых окон, вызываемых из приложений на Автолиспе, показаны приемы использования сценариев, слайдов и слайд-фильмов. Изложение иллюстрируется большим количеством разнообразных примеров, как для 12-й версии Автокада, так и для более младших. Представлена аннотация Автокада версии 13 и некоторых приложений для архитектуры, строительства и машиностроения.

К $\frac{2404000000-013}{Г70(03)-96}$ Без объявл.

Учебно-справочное издание

Юрий Андреевич Кречко

AutoCAD: программирование и адаптация.

Редактор О. А. Голубев
Макет О. А. Кузьминовой
Обложка Н. В. Дмитриевой
Корректор В. С. Кустов

Лицензия ЛР N 070109 от 29.08.91. Подписано в печать 26.06.95.
Формат 60x84/16. Бум. офс. Печать офс. Гарнитура Таймс.
Усл. печ. л. 13.95. Уч.-изд. л. 8.1. Доп. тираж 3 000 экз. Заказ 355.

Акционерное общество "ДИАЛОГ-МИФИ"
115409, Москва, ул. Москворечье, 31, корп. 2

Подольская типография
142110, г. Подольск, Московская обл., ул. Кирова, 25

ISBN 5-86404-060-6

Предисловие

Автокад является не только графическим редактором, но и мощной средой программирования. В 11-ой версии этого программного продукта, частично, и в 12-й версии в полном объеме появилась возможность разрабатывать приложения на Автолиспе, Си как в защищенном, так и в реальном режимах, формировать связи атрибутов блоков с базами данных для различных СУБД (Paradox, dBASE), программно модифицировать строки и разделы в пользовательских меню, создавать диалоговые окна в приложениях на языках Си и Автолиспе. Это открывает широкие возможности для разработки специализированных САПР, ориентированных на конкретные приложения. Программы и меню, составленные для более ранних версий Автокада, после незначительных изменений могут широко использоваться и в 11-й и в 12-й версиях. Необходимость внесения изменений связана с тем, что некоторые команды более поздних версий претерпели незначительные изменения и их протокол выполнения либо должен быть изменен, либо соответствующие системные переменные должны быть настроены надлежащим образом. Данное пособие может использоваться как для работы с 12-й версией Автокада, так и с более ранними версиями, разумеется, с отдельными ограничениями.

1

Автолисп - базовый язык программирования в Автокаде

Все или почти все, что мы делаем в среде Автокада вручную, можно реализовать программно. Для этого в Автокаде предусмотрено несколько механизмов и средств. Это, во-первых, средства создания сценариев и, во-вторых, использование языков программирования. В Автокаде до 11-ой версии таким языком являлся Автолисп. Поскольку он зарекомендовал себя удобным и практичным средством программирования, Автолисп оставлен и в 12-й версии, и разработчики предполагают, что совместное использование Автолиспа с языком Си сохранится и в дальнейших версиях. Более того, перед тем, как приступить к разработке приложений на языке Си, настоятельно рекомендуется ознакомиться с принципами и методами создания программ на Автолиспе. Поэтому наше изложение мы начинаем с языка Автолисп.

Будучи частью Автокада, Автолисп позволяет оперировать с переменными различных типов и передавать их значения командам Автокада при вводе данных. При ответах на запросы команд Автокада можно также использовать выражения Автолиспа, в которых могут выполняться различные арифметические и условные операции над числовыми значениями и значениями определенных переменных.

Кроме средств выполнения вычислений Автолисп содержит в себе функции, предоставляющие средства доступа к графической базе данных текущего чертежа Автокада. Автолисп позволяет также управлять графическим редактором Автокада и обращаться к собственным командам Автокада. Программы Автолиспа позволяют создавать настроенные на конкретную предметную область применения функции, включающие в себя запросы к пользователю (диалог), возможность выбора по условию из нескольких вариантов или использования значений по умолчанию. Хотя макроопределения, которые вы можете создавать при написании меню Автокада, могут быть достаточно сложными и мощными, без Автолиспа они представляют собой всего лишь комбинации стандартных команд Автокада. Включением в макроопределения меню функций Автолиспа можно превратить меню Автокада в интеллектуальное средство автоматизации проектирования.

Автолисп позволяет:

- использовать переменные и выражения при ответах на запросы команд Автокада;

- читать и писать внешние файлы, таким образом обмениваясь с внешними программами, которые можно запускать из Автокада;
- создавать различные функции и новые команды Автокада, настраивая и расширяя таким образом графические возможности Автокада;
- осуществлять программный доступ (считывание и изменение) к данным, относящимся к объектам рисунка, а также таблицам Автокада, в которых хранится информация о блоках, слоях, видах, стилях и типах линий;
- осуществлять программное управление графическим экраном Автокада и вводом/выводом с различных устройств.

1.1. Что такое Автолисп

Автолисп - это созданный специально для Автокада диалект Лиспа, полученный в результате изменения языка CommonLISP.

Язык LISP разработан в 1961 году американским ученым Дж. Маккарти и является родоначальником функциональных языков. Автолисп относится к так называемым функциональным языкам, т. е. к языкам, в основу которых положено понятие функции (в отличие от обычных операторных языков программирования - ФОРТРАНа, Паскаля, Си и др.). К таким языкам относится также, например, язык Пролог. Все вычисления, преобразования и управления программы в функциональных языках осуществляются с помощью элементарных (встроенных) функций или функций, определяемых программистом при написании программы. Программа в целом является суперпозицией некоторых функций и, в свою очередь, может быть использована как функция другими программами.

Язык Лисп идеально подошел для Автокада, "передав" Автолиспу свои очень удобные средства работы с глубоко структурированной информацией. Автолисп - это средство, позволяющее программно работать с объектами Автокада, справочными таблицами, считывать и записывать файлы из Автокада. Автолисп можно считать прямым окном внутрь Автокада. Кроме того, Автолисп очень прост в изучении и весьма гибок.

С введением в Автокад версии 11 Средства разработки приложений (СРП) появилась возможность писать приложения и на других языках. Однако фирма Autodesk и далее намеревается поддерживать Автолисп, рассматривая СРП как дополнение к нему.

1.2. Установка Автолиспа

Автолисп ранних версий поставлялся с каждой копией Автокада, имеющей модуль ADE-3. Для операционных систем PC-DOS/MS-DOS файл acad.ovl на дистрибутивном диске системы Автокад является оверлейным файлом интерпретатора Автолиспа.

Один из дистрибутивных дисков содержит файл с именем readme.doc. Следует ознакомиться с содержимым этого файла, поскольку он содержит все последние изменения и дополнения к документации по Автокаду и Автолиспу.

Для работы в операционных системах PC-DOS/MS-DOS с 10-й или 11-й версией Автокада имеется дополнительная версия Автолиспа, так называемый расширенный Автолисп, содержащийся в файлах acadlx.ovl, extlisp.exe и remlisp.exe.

Обычный (не расширенный) Автолисп может работать на любой машине, на которой работает Автокад версии 10 или 11 с модулем ADE-3. Для работы расширенного Автолиспа необходимо также, чтобы компьютер был построен на базе микропроцессора типа Intel 80286 или 80386 (не 8086 или 8088). Для работы расширенного Автолиспа желательно, чтобы ваш компьютер имел не менее 512 Кбайт расширенной памяти типа IBM AT extended memory (не типа Lotus/Intel/Microsoft expanded memory), не занятой под другие цели, например под виртуальный (электронный) диск.

При настройке Автокада одним из рабочих параметров является отключение Автолиспа по желанию пользователя. Кроме того, в операционных системах PC-DOS/MS-DOS конфигуратор 10-й и 11-й версий дает возможность отключения расширенного Автолиспа. (Если вы не работаете с Автолиспом, на этот вопрос можно безболезненно отвечать "Нет".)

В операционных системах PC-DOS/MS-DOS для Автолиспа должна быть выделена определенная часть свободной оперативной памяти, объявленная как память-стек и память-хип, при помощи задания значений переменных среды LISPHEAP и LISPSTACK. В случае одновременной работы с каким-либо другим прикладным пакетом, например Автокад AEC, следует ознакомиться с содержащимися в его описании рекомендациями по установке значений переменных среды LISPHEAP и LISPSTACK. Размером области расширенной памяти, которая будет использоваться расширенным Автолиспом, можно управлять с помощью переменной среды LISPXMEM (если переменная LISPXMEM не задана, расширенный Автолисп использует всю оперативную память, кроме занятой виртуальным диском). Расширенный Автолисп сообщает Автокаду, какую область памяти он использует, тем самым делая эту об-

ласть памяти недоступной для использования самим Автокадом. Расширенный Автолисп не использует хип.

Расширенный Автолисп действует как самостоятельная программа с именем EXTLISP, которую необходимо запустить перед запуском Автокада. EXTLISP является резидентной программой, которая может размещаться в расширенной памяти и связывается с Автокадом через оверлейный файл acadxl.ovl. Для автоматического запуска программы EXTLISP ее вызов можно поместить в файл настроек системы Автокад, acad.bat или даже (если вы ни с чем, кроме Автокада, на машине не работаете и вас устраивает, что EXTLISP будет автоматически загружаться в память при включении компьютера) в файл autoexec.bat.

Программа EXTLISP может быть удалена из памяти после окончания сеанса работы с Автокадом. Для этого в ответ на подсказку DOS следует ввести команду REMLISP, вызывающую на выполнение программу с тем же именем. При этом расширенный Автолисп выгружается из оперативной памяти.

Все, что написано в этом пункте, совершенно справедливо для Автолиспа, используемого для Автокада 11, предназначенного для IBM PC 286. В комплект Автокада 386 11-й и 12-й версий не входит EXTLISP, и настройками Автолиспа там заниматься особенно не приходится.

1.3. Работа с Автолиспом с командной строки Автокада

К Автолиспу можно обращаться прямо с командной строки Автокада. Когда вы вводите какую-то группу букв, интерпретатор командной строки пытается найти введенное вами слово в списке команд Автокада. Если введенного слова нет в списке команд, то выдается сообщение об ошибке. Напомним лишний раз, что команда Автокада не может содержать пробелов и нажатие клавиши “ПРОБЕЛ” воспринимается так же, как нажатие клавиши “RETURN”.

Если первым символом вы введете в командной строке круглую открывающую скобку, то интерпретатор командной строки Автокада переходит в специальный режим - режим ввода выражения Автолиспа. Выход из этого режима осуществляется при вводе скобки, закрывающей вводимое выражение (оно может быть сложным и содержать вложенные выражения). Работа с клавиатурой в этом режиме имеет несколько особенностей. Так, нажатие клавиши “ПРОБЕЛ” не рассматривается больше как терминатор ввода, поскольку любое выражение Автолиспа содержит пробелы. Кроме того, выражение Автолиспа может занимать несколько строчек и поэтому передается вычислителю Автолиспа только

тогда, когда он завершит ввод последней правой скобки. Поэтому, если при вводе вами какого-то выражения Автокад выдаст сообщение "1>", не беспокойтесь - это просто подсказка, которая говорит о том, что для правильного завершения ввода функции или выражения необходима еще одна правая круглая скобка - ")". Вообще, ответ Автолиспа типа "n>" (n - целое число) говорит о том, что интерпретатору Автолиспа требуется n закрывающих скобок или кавычек для строковых констант (каждая открывающая скобка или кавычка должна иметь свою закрывающую скобку или кавычку). Введите скобку, нажмите "RETURN", затем "CTRL C" и попытайтесь набрать пример еще раз, внимательно следя за балансом скобок и кавычек. Если добавление скобок в ответ на этот запрос не помогает, введите кавычку, а после нее - соответствующее количество скобок, потому что если вы забыли кавычку, то все последующие данные (в том числе и круглые скобки) интерпретируются как текст и не распознаются вычислителем Автолиспа. Самое трудное в Автолиспе - это отслеживать правильные пары скобок и кавычек ("") для строковых констант. Для этого мы рекомендуем при составлении программ использовать текстовые редакторы с возможностью отслеживания пар скобок. Это может быть, например, Norton Editor фирмы Symantec.

Повторяя приводимые ниже упражнения на Автокаде, вы не увидите на экране строки "Автолисп возвращает:", однако все, что написано после этой строки, должно появиться на экране.

Автолисп может быть полезен не только как средство программирования, но и просто как калькулятор, которым можно воспользоваться в любой команде Автокада в ответ на запрос ввести какое-то значение или координаты точки. Парадокс, однако, заключается в том, что работа с этим "калькулятором" предполагает достаточно свободное владение Автолиспом и умение на нем программировать. Исходя из этого, мы надеемся, что, изучив эту главу и написав несколько небольших программ, вы начнете применять Автолисп в своей повседневной деятельности и самостоятельно найдете удобные для себя приемы и методы использования Автолиспа при интерактивной работе с графическим редактором Автокада.

1.4. Типы данных и переменные

Автолисп использует переменные. Переменная - это просто метка, используемая для обращения к изменяемой величине. Вам приходится сталкиваться с переменными каждый день. Например, получаемая вами ЗАРПЛАТА формируется из ОКЛАДА, из которого вычитается НАЛОГ, и ПРЕМИИ ЗАРПЛАТА, ОКЛАД, НАЛОГ и ПРЕМИЯ - переменные.

Автолисп позволяет поставить в соответствие имя переменной и ее значение, которое будет потом использоваться в Автолиспе в выражениях и функциях.

1.5. Типы данных Автолиспа

Благодаря Автолиспу вы имеете доступ к стандартным системным переменным (о них рассказывается в п. 1.4.3) и можете создавать переменные Автолиспа. Как и любой другой язык программирования, Автолисп при работе с переменной должен знать, к какому типу данных она относится: типом данных определяются операции, которые можно проделать с переменной (например, одно число поделить на другое число, но нельзя поделить одну строку на другую).

Понятие типа данных - ключевое понятие в программировании. "Качество" языка программирования, т. е. его потенциальные возможности, определяются во многом тем, какие типы данных им поддерживаются. Перечислим сначала основные типы данных, которые есть в любом языке программирования, в том числе и в Автолиспе:

- списки;
- символы;
- строковые константы;
- действительные числа;
- целые числа;
- дескрипторы файлов;
- "имена" примитивов Автокада;
- наборы Автокада;
- встроенные функции (subr);
- внешние функции (функции СРП).

В программировании на языках семейства Лисп используются символы и построенные на них символьные структуры. В соответствии со словарным определением, символом является опознавательный знак, условное обозначение, художественный образ, обозначающий какую-либо мысль, идею и т. п. В нашем случае понятие символа используется в более узком и точном смысле: под символом подразумевается запись или обозначение.

Символ - это имя, состоящее из букв, цифр и специальных знаков за исключением () - круглых скобок, . - точки, ' - апострофа, " - кавычек, ; - точки с запятой. Имя символа обрывается круглыми скобками, апострофом, кавычками, точкой с запятой, пробелом или концом строки.

Список представляет собой упорядоченную последовательность, элементами которой являются атомы или списки. Списки заключаются

в круглые скобки, элементы списка разделяются пробелами. Список всегда начинается с открывающейся скобки и заканчивается закрывающейся. Атом - это неделимый элемент списка. Понятия списка и атома являются ключевыми понятиями языков семейства Лисп. Любые другие типы данных являются либо атомами, либо списками.

Примеры списков:

```
(12.6 45.7 77.8)
("CAT" "EATS" "MOUSE")
(1 (12 6.78) ("Иван" "Грозный"))
(* 2 5)
(setq point '(1.0 5.0 6.7))
```

Примеры атомов:

```
12.45
"True Love"
()
```

Строковая константа выглядит как набор печатных символов, заключенных в кавычки.

Целые числа в Автолиспе могут быть положительными или отрицательными (без дробей и десятичной точки). Для определения того, включен какой-то режим или нет, Автокад часто использует значения 0 и 1, например если режим ORTHO (ОРТО) включен, то системная переменная ORTHOMODE установлена в единицу. Заметим, что целые числа представлены в машине двумя байтами и не могут поэтому выходить за диапазон (-32768, 32767).

Действительные числа - это положительные или отрицательные числа с десятичной точкой. В отличие от многих других языков программирования в Автолиспе не разрешается начинать или заканчивать число десятичной точкой. Это связано с существованием в Автолиспе специфического типа данных - точечной пары, разделителем которой является точка. Поэтому, если значение меньше единицы, вы должны явно указать 0 перед десятичной точкой (0.123), иначе Автолисп воспримет ошибку (error: invalid dotted pair (ошибка: неверная точечная пара). В качестве примеров действительных системных переменных можно указать переменные FILLETRAD, LTSCALE и AREA. Диапазон представления действительных чисел зависит от типа вашего компьютера и в любом случае достаточно велик. Правильно записанными действительными числами являются числа 2.12, 3.11592652543, -92722.121344 и 1.23544E+17.

Требование принадлежности переменной к какому-либо типу данных - бич для программиста, и вообще требование, чуждое обыденному человеческому мышлению. Ведь тип данных - чистая абстракция, соглашение - уступка вычислительной технике, которая пока еще не слиш-

ком-то совершенна. Человек не оперирует понятием "тип данных", для него данные не имеют фиксированного типа (как говорится, с какой точки зрения посмотреть), а скорее имеют все типы одновременно: написанное на бумаге число - это одновременно и группа символов, и число, и вообще что угодно, например художественное произведение абстрактной живописи. Мы уже говорили, что Автолисп был создан как язык, предназначенный для разработок в области искусственного интеллекта, и проблема типа данных в нем в какой-то степени решена. Переменные в Автолиспе не надо описывать - тип данных Автолисп определяет автоматически при присвоении переменной значения (те, кто программировал на ФОРТРАНе или Паскале, оценят это).

Как уже ясно из примеров, списки - идеальное средство работы со сложным образом организованной геометрической информацией. Точка, например, не что иное, как список трех действительных чисел. Отрезок - список двух точек. Полилиния - список многих вершин (как видим, полилиния - геометрический объект, органически вытекающий из "идеологии" описания данных Автолиспа). Практически любую структурированную информацию можно представить в виде списка, что и делается в Автолиспе. Более того, любая определяемая пользователем функция Автолиспа (это соответствует подпрограмме в операторных языках программирования), вообще говоря, тоже список.

Типы данных Автолиспа с помощью функции TYPE идентифицируются следующим образом:

Идентификатор	*Тип данных*
INT	Целые величины
REAL	Числа с плавающей точкой
STR	Строковые константы
FILE	Дескрипторы файлов
SYM	Символы
LIST	Списки и функции пользователя
SUBR	Внутренние функции Автолиспа
EXSUBR	Внешние функции
ENAME	Имена примитивов Автокада
PICKSET	Наборы примитивов Автокада
PAGETB	Таблица диспетчера страниц

Итак, переменная в Автолиспе может быть атомом (целого, вещественного или строкового типа), а может быть списком. Приведем пример, в котором формируется список координат точки.

```
Команда:(setq a '(100 200 300))
Автолисп возвращает: (100 200 300)
```

В этом примере создается переменная А, которая является списком. В Автолиспе имеются мощные средства работы со списками. Позже (см. п. 1.16) мы рассмотрим средства работы со списками Автолиспа более подробно.

1.6. Переменные Автолиспа

Присваивая значение символу (идентификатору, имени переменной), вы автоматически создаете переменную (при этом Автолисп фиксирует ее тип). При создании переменной (а точнее, при присвоении идентификатору переменной какого-то значения) Автолисп автоматически назначает ей тип. Переменные Автолиспа совершенно независимы от системных переменных Автокада и могут повторять их по именам. При каждом обращении к переменной Автолисп использует то значение переменной, которое было установлено последним. Имя переменной может состоять из любых печатных символов (в том числе и цифр), однако не должно включать в себя зарезервированные Автолиспом символы, поскольку их Автолисп интерпретирует специальным образом. Не рекомендуется использовать для имен переменных следующие символы:

- Зарезервированные символы:
  ., ', ", ;, (,) или пробел.
- Знаки операций Автолиспа:
  ~, *, =, >, <, +, -, /.

Нельзя также использовать в качестве имен переменных имена функций Автолиспа. Все имена функций Автолиспа, а также созданные пользователем имена переменных хранятся в переменной Автолиспа ATOMLIST. Выведите на экран содержимое переменной ATOMLIST и вы увидите, какие имена вы не можете использовать в своей программе. (Это можно сделать вводом с командной строки Автокада команды !ATOMLIST.) Некоторые из этих идентификаторов используются исключительно вычислителем Автолиспа и не могут быть вами использованы, остальные описаны в "Руководстве по программированию на Автолиспе" из комплекта документации, прилагаемой к Атокаду, к которому мы рекомендуем вам обращаться за справочной информацией о встроенных функциях Автолиспа.

Заметим, что Автолисп не различает строчные и прописные буквы. Поскольку имя переменной длиной более шести символов требует для хранения больше памяти, мы не рекомендуем вам превышать этот предел. Не следует также начинать имя переменной с цифры.

Имена переменных		
неверные		*верные*
123	- целое число	PT1
10.5	- действительная константа	txt
ANGLE	- конфликтует с функцией ANGLE	ANGL
A(1)	- содержит запрещенные символы	A1
OLD SUM	- содержит пробел	OLDSUM

1.7. Системные переменные Автокада

Среда графического редактора Автокада позволяет изменять режимы, переопределять лимиты, менять размеры прицелов объектной привязки и выбора объектов, получать информацию о различных установках графического редактора.

Установленные вами режимы сохраняются до тех пор, пока вы их не переопределите. Сохраняются не только общие настройки (например, размер прицела выбора объектов), но и настройки, относящиеся к способу работы с чертежом (единицы измерения, режим отрисовки маркеров и т. п.).

Все настройки графического редактора управляются так называемыми системными переменными Автокада. Когда вы изменяете установки, например включаете или выключаете режим ORTHO (ОРТО), изменяете режим привязки и т. п., Автокад сохраняет только что установленный режим в соответствующей системной переменной. Системные переменные, таким образом, формируются в процессе работы с графическим редактором. Фиксировано не только имя, но и тип системной переменной (в вещественную переменную, например, нельзя записать строку).

Системные переменные делятся на модифицируемые пользователем и защищенные. Защищенную переменную нельзя изменять ни программно, ни при помощи команды SETVAR (УСТПЕРЕМ). Она может быть изменена только при выполнении определенной команды. Например, при создании нового примитива за границами зоны чертежа, содержавшей ранее созданные примитивы, и последующем выполнении команды ПОКАЖИ Границы системные переменные EXTMIN и EXTMAX изменятся и примут новые значения. Те же переменные, которые не имеют такой защиты, могут быть вами изменены прямо в процессе выполнения той или иной команды путем использования

команды 'SETVAR ('УСТПЕРЕМ) в прозрачном режиме. Меняя значения системных переменных, вы можете управлять многими (правда, не всеми) режимами графического редактора Автокада. Системные переменные также можно разделить по месту их хранения: некоторые из них записываются по окончании сеанса редактирования в файл конфигурации системы, остальные (таких большинство) - в текущий чертеж. Таким образом, файл чертежа Автокада - это не просто описание геометрических объектов, а еще и полное описание режимов работы и настроек Автокада. Некоторые переменные никуда не записываются, потому что никогда не меняются - например, номер версии Автокада (переменная ACADVER).

При помощи команды SETVAR (УСТПЕРЕМ) вы можете в любой момент посмотреть текущие установки системных переменных:

```
Command:SETVAR
Variable name or ?:?
```

```
Команда:УСТПЕРЕМ
Имя переменной или ?:?
```

Системные переменные Автокада могут быть считаны или установлены из Автолиспа, для этого существуют встроенные функции Автолиспа getvar и setvar. (Не надо путать функцию setvar Автолиспа и команду SETVAR Автокада - это совершенно разные вещи.) Для того чтобы в программах на Автолиспе не изменять установки системных переменных, используемые Автокадом по умолчанию, необходимо считывать (при помощи функции getvar) значение изменяемой системной переменной, сохранять старое значение и затем (при помощи функции setvar) восстанавливать его перед выходом из программы. В макроопределениях также следует восстанавливать старые значения системных переменных - это хороший стиль программирования на Автолиспе, он поможет вам застраховаться от ошибок.

Список системных переменных приведен в приложении А.

1.8. Выражения Автолиспа

Если с точки зрения Автолиспа все, что заключено в круглые скобки, считается списком, то как Автолисп отличает данные от управляющих конструкций - объект, который оперирует данными, от самих данных? Дело в том, что любая управляющая конструкция Автолиспа содержит объекты специального типа - функции. Функции Автолиспа могут быть встроенными или определенными пользователем. Встроенная функция Автолиспа (subr) - это тип данных, являющийся на самом деле ссылкой на некую подпрограмму, которая выполняет над передаваемыми параметрами некоторые действия. Функция, определенная

пользователем, - это обычный сложный список, но все равно он в конечном итоге содержит встроенные функции Автолиспа. Для того чтобы Автолиспу было проще искать среди элементов списка функции, их принято писать на первом месте.

Список, в котором первым элементом является имя функции, будем называть выражением. Любая функция Автолиспа состоит из выражений и сама является выражением. Перечислим основные свойства выражений (многие из этих свойств обусловлены просто тем, что любое выражение - это список).

- Каждая открывающая круглая скобка должна иметь закрывающую.
- Сразу после открывающей круглой скобки должен стоять идентификатор операции, выполняемой при вычислении выражения (имя функции).
- Каждое выражение вычисляется (выполняется), и результат возвращается. Результатом может быть нуль (nil) или результат вычисления последнего подвыражения.
- С логической точки зрения любое возвращаемое значение либо истинно, либо ложно. Если значение выражения вычислено быть не может и возвращается пустое место (nil), то оно считается ложным. Если значение вычисляется, то выражение считается истинным - не-пусто (non-nil).

Выражение Автолиспа имеет вид

```
(функция аргумент1 аргумент2.... аргументN)
```

Здесь функция - имя операции, которая должна быть выполнена. К операциям относятся в том числе и арифметические операции (сложение, вычитание, умножение, деление), а также операция присвоения. Записывать имя функции нужно сразу же после открывающей скобки без пробела. Аргументы представляют собой средство передачи значений функции. Аргументами могут быть переменные, константы или выражения. Число аргументов функции может быть переменным. Например, операция перемножения трех чисел запишется так: (*2 5 8).

Аргументы могут быть обязательными и необязательными. Если при создании функции было определено, что ее аргументы являются обязательными, то Автолисп выдаст ошибку при попытке вызвать функцию без указания этого аргумента. Однако аргументы могут быть и необязательными, в этом случае их при вызове функции можно не передавать. Следует также внимательно следить за типом аргументов: если функция производит операции над числами, то ей нельзя передать строку, и наоборот. Иначе говоря, если функция (или команда Автокада) должна принять в качестве аргумента строку, все символы, введенные в ответ на запрос, считаются элементами строки a priori. На первый взгляд это не позволяет передать Автокаду выражение Автолиспа в ответ на запрос, требующий ввода строки (например, при использовании

команды TEXT). Указанную трудность помогает преодолеть системная переменная Автокада TEXTEVAL. Установка этой переменной в единицу (оценка текста включена) приводит к тому, что Автокад передает выражение в Автолисп и печатает возвращаемый результат, а не само выражение. Другими словами, если оценка текста включена, то в ответ на запрос Автокада можно вводить не только сам ответ, но и выражение на Автолиспе для его вычисления, используя круглые скобки или восклицательный знак.

Выражение анализируется Автолиспом слева направо, пока не встретится закрывающая или открывающая скобка. Если встречается закрывающая скобка, то Автолисп завершает анализ выражения, выполняет функцию и передает ее значение на более старший уровень вложенности или в Автокад. Если встречается открывающая скобка, Автолисп переходит к анализу выражения более младшего уровня вложенности и, пока не завершит его анализ, не перейдет к дальнейшему анализу выражения предыдущего уровня. Таким образом, порядок анализа выражений в Автолиспе задан жестко раз и навсегда, что исключает путаницу.

При описании функций необязательные параметры мы будем заключать в квадратные скобки.

1.9. Функции присвоения

Присвоение переменным значений осуществляется в Автолиспе не так, как в других языках программирования. Для присвоения переменным значений в Автолиспе имеется специальная функция SETQ, которая присваивает переменной нужное нам значение, т. е. заносит указанное нами значение в ячейку с именем переменной. Алгебраическая запись присвоения - x = 3. В Автолиспе то же действие записывается как (setq x 3). Это связано с тем, что, как уже отмечалось, Автолисп - функциональный язык и в нем нет операторов (в том числе и оператора присвоения), а есть функции, производящие определенные действия над своими параметрами.

После того как вы присвоили некоторой переменной значение, вы можете в любое время посмотреть ее значение с командной строки Автокада. Восклицательный знак "!", введенный с командной строки, сообщает интерпретатору командной строки Автокада, что следующий за ним идентификатор является именем переменной Автолиспа. Встретив восклицательный знак, интерпретатор командной строки передает следующее за ним имя Автолиспу, который возвращает обратно значение переменной с этим именем.

Что такое SET - понятно: “положить”, а что означает SETQ? Загадочная буква “q” в функции присвоения SETQ происходит от сокращения английского слова “quote” - буквально “цитировать”. Функция SETQ является мнемоническим сокращением английского словосочетания “SET by Quote” - “присвоить по ссылке”. Прежде чем разобраться, что это означает, нужно понять, как хранятся в машине значения переменной. Дело в том, что переменная и ее значение не одно и то же. На первый взгляд такое утверждение кажется тривиальным, однако когда мы говорим “переменная”, то чаще всего подразумеваем “значение, которое хранится в переменной”. Специфика Автолиспа заключается в том, что в переменной может храниться не только значение, но и целое выражение, а если быть точным, то его адрес. Функция SETQ позволяет об этом не задумываться и считает, что мы обращаемся к значению переменной. В одной функции SETQ можно осуществить несколько присваиваний:

```
(setq a "Новый" b 3.14 c '(34.0 77.6 55.4)
```

Однако в Автолиспе есть также функция SET, которая считает, что мы обращаемся к самой переменной. Прежде чем проиллюстрировать сказанное на примерах, отметим, что для обращения к значению переменной в Автолиспе существует специальная функция ’ (quote выражение), которая возвращает само выражение, не выполняя его (фактически возвращает ссылку на выражение). Эта функция очень часто используется в Автолиспе и поэтому имеет сокращенную запись: ‘выражение. Так, (quote A) эквивалентно ‘A.

Теперь продемонстрируем работу функций SETQ и SET:

```
(set 'A 5.0)
```

Возвращает 5.0 и присваивает это значение A.

Эта запись эквивалентна записи (setq a 5.0)

```
(set 'B '(A + C))
```

Возвращает выражение (A+C) и присваивает его B.

Итак, в переменной хранится выражение! Причем это именно выражение, а не текстовая константа - изменяя значения переменных A и C, мы изменяем тем самым значение переменной B. Эта запись эквивалентна записи

```
(setq B '(A + C))
```

Обратившись к специальной функции EVAL (“evaluate” - “оценивать”), можно в любой момент вычислить выражение B. Предположим, что перед выполнением предыдущей строки мы положили (setq A 3) и (setq C 6). Тогда

```
(eval B)
```

Возвращает 9.

Если две переменные "вложены" друг в друга, то к младшей можно обращаться через старшую, т. е. косвенно. Выполните следующую последовательность функций:

(set 'B 'A)

Устанавливает A в B и возвращает A.

(set B 640)

Устанавливает A в 640 и возвращает 640.

По сути, в функции (set B 640) сначала вычисляется значение B, потом результату этого вычисления присваивается 640.

1.10. Математические функции

Автолисп включает в себя достаточно широкий набор встроенных функций, позволяющих производить математические вычисления. Аргументами математических функций являются числа, которые могут быть как целыми, так и вещественными. Если все аргументы - целые числа, то результат операции также будет целым числом, а любая дробная часть будет опущена. Если хотя бы один аргумент - вещественное число, то результат операции будет вещественным числом. (Следует отметить, например, что число 2.0 считается вещественным, поскольку в нем присутствует десятичная точка.) К математическим функциям относятся:

Функция	*Назначение*
(+ число1 число2 ...)	Возвращает сумму всех аргументов
(- число1 число2 ...)	Вычитает число2 из числа1 и возвращает разность. Если задано более двух аргументов, то из первого аргумента вычитается сумма всех остальных. Если задан один аргумент, то он вычитается из нуля (инвертируется его знак)
(* число1 число2 ...)	Возвращает произведение всех чисел
(/ число1 число2 ...)	Делит число1 на число2 и возвращает частное. Если задано более двух аргументов, то первое число делится на произведение всех остальных
(1+ число)	Увеличение целого или действительного аргумента на единицу
(1- число)	Уменьшение целого или действительного аргумента на единицу

(atan число1 [число2])	Если не задано число2, то возвращает арктангенс переменной число1 в радианах, область допустимых значений - [pi,-pi] радиан. Если заданы оба числа, то возвращается арктангенс переменной число1/число2 в радианах. Если число2 - нуль, то в зависимости от знака переменной число1 возвращается + или - 1.570796 радиан (+90 или -90)
(abs число)	Вычисление абсолютного значения действительного или целого числа
(sin число)	Возвращает значение синуса угла, заданного аргументом в радианах
(cos число)	Возвращает значение косинуса угла, заданного аргументом в радианах
(exp степень)	Вычисляет значение экспоненциальной функции с основанием е и аргументом, равным числу
(expt основание степень)	Вычисляет значение экспоненциальной функции с указанными основанием и степенью
(gcd число1 число2)	Возвращает наибольший общий делитель (Greatest Common Denominator) указанных аргументов
(log число)	Возвращает натуральный логарифм аргумента
(max число1 число2 ...)	Возвращает наибольший аргумент. Эта функция в отличие от многих других не изменяет типов чисел
(min число1 число2 ...)	Возвращает наименьший аргумент. Не изменяет типов чисел
(rem число1 число2 ...)	Возвращает остаток (Remainder) от деления переменной число1 на переменную число2 (математическая запись: число1 mod число2)
(sqrt число)	Извлекает квадратный корень из аргумента. Возвращаемый результат всегда вещественный
(lsh число1 числобит)	Возвращает побитовый сдвиг числа1 на число числобит. Оба аргумента должны быть целыми, результат тоже будет целым. Если числобит положительно, то число1 сдвигается влево (оно умножается на 2 в степени числобит), если отрицательно - сдвигается вправо (делится на 2

	степени числобит нацело с отбрасыванием остатка). В каждом случае "нулевые" биты добавляются, а сдвигаемые биты сбрасываются. Если "единичный" бит сдвигается в высший, 16-й разряд целого числа, знак числа меняется
pi	Константа, равная 3.1415926

Функции Автолиспа могут быть, за очень редким исключением, вложенными. Например, чтобы вычислить корень квадратный из суммы квадратов двух чисел, можно записать

```
(setq a (sqrt (+ (* c c) (* b b))))
```

1.11. Работа со строками, функции преобразования

Среди встроенных функций Автолиспа, работающих со строками, можно найти практически все, что имеется в любом развитом языке высокого уровня:

Функция	*Назначение*
(itoa целое)	Возвращает результат преобразования целого числа в строковую константу
(atoi строка)	Возвращает преобразование строковой константы в целое число
(atof строка)	Возвращает преобразование строковой константы в действительное число
(strcase строка признак)	Переводит все символы аргумента строка в нижний регистр, если признак не nil, если признак опущен или равен nil - в верхний
(substr строка целое1 целое2)	Возвращает подстроку аргумента строка, начинающегося с символа целое1 и длиной целое2. Первый символ строки имеет номер 1
(strlen строка)	Возвращает длину в символах аргумента строка
(strcat строка1 строка2 ...)	Осуществляет сцепление аргументов строка
(chr число)	Возвращает результат преобразования целого числа в символьный код ASCII, причем результатом является строковая константа. Например, (chr 65) возвратит символ (строковую константу) "A" (подобным образом действует функция CHR$ в языке Бейсик)

(ascii строка)	Возвращает преобразование одного символа, указываемого в виде строковой константы, в код ASCII

В любой момент можно преобразовать целое число в действительное и наоборот:

Функция	*Назначение*
(fix число)	Возвращает результат преобразования действительного числа в целое. Преобразование осуществляется усечением (отбрасыванием) дробной части
(float число)	Возвращает результат преобразования целого числа в действительное
(rtos число [режим [точность]])	Возвращает текстовую строку, которая представляет число (вещественная величина) в соответствии со значениями режима, точности и системной переменной Автокада DIMZIN. Режим и точность - целые числа, которые назначают режим представления числа и его точность: *Режим RTOS* — *Описание* 1 — Научный 2 — Десятичный 3 — Инженерный (футы и десятичные дюймы) 4 — Архитектурный (футы и дробные дюймы) 5 — Произвольные дробные части Аргументы режим и точность соответствуют системным переменным Автокада LUNITS и LUPREC. Если эти аргументы отсутствуют, то используются текущие значения LUNITS и LUPREC
(type элемент)	Возвращает тип элемента Используя функцию type (о ней уже говорилось выше), можно определить тип элемента. Пример использования функции TYPE: `(defun isint (a)` `  (if (= (type a) 'INT)   ; является ли а целым?` `    T                     ; если да - возвращаем ИСТИНУ` `    nil                   ; если нет - возвращаем nil` `  )` `)`

1.12. Использование функций GET для ввода данных

Для ввода данных с клавиатуры в Автолиспе существует семейство функций GET. Для всех основных типов данных есть своя функция GET. Все аргументы функций этого семейства необязательны. В качестве первого аргумента иногда выступает необязательная двумерная точка в текущей ПСК. Все функции GET могут иметь в качестве аргумента произвольную строковую константу, в которой может содержаться текст запроса или какая-то подсказка, выводимая при запросе пользователю ввести какие-то данные. Все функции GET ожидают ответа пользователя, т. е. приостанавливают выполнение программы до тех пор, пока не будет осуществлен ввод (нажата клавиша "RETURN"). Ввод может быть осуществлен как с клавиатуры, так и при помощи устройства указания. Вводя точки с экрана, вы можете захотеть, чтобы в процессе перемещения курсора по экрану Автокад показывал "резиновую" линию. Это позволяют делать практически все функции семейства GET. Все вводимые данные автоматически преобразуются в нужный тип данных. В ответ на запрос функций семейства GET нельзя вводить выражение Автолиспа: это приведет к ошибке и выводу сообщения "Can't reenter AutoLISP" ("Не могу войти в Автолисп повторно"). В макроопределениях меню функции GET нужно выделять обратной косой чертой с тем, чтобы Автокад обеспечил паузы для ввода данных. Вот список этих функций:

(getangle [точка] [текст запроса-подсказки])

Возвращает угол в радианах между задаваемым пользователем вектором и положительным направлением оси X в текущей плоскости построений. Функция всегда возвращает угол в радианах вне зависимости от текущей установки переменных. Начальная точка вектора может быть определена первым аргументом функции. Вторую точку вектора можно указать на экране мышью, при этом Автокад нарисует "резиновую" линию от точки, указанной первым аргументом, до текущего положения курсора. Если первый аргумент опущен, Автокад потребует ввода двух точек. Не запрещается указывать трехмерные точки (см. также функцию GETORIENT).

(getcorner точка ["текст запроса-подсказки"])

Эта функция, так же как и getpoint, возвращает координаты указанной пользователем точки в текущей ПСК. Отличие от getpoint заключается в том, что функция getcorner строит "резиновую" рамку при передвижении курсора по экрану, первый угол которой может быть определен первым аргументом функции. В этой функции первый аргумент

обязателен. Если вводится трехмерная точка, то координата Z игнорируется (см. функции getpoint и getdist).

(getdist [точка] [текст запроса-подсказки])

Какими бы ни были текущие единицы измерения (например, футы), эта функция всегда возвращает действительное число. Если расстояние указывается с клавиатуры, функция тестирует значения системной переменной FLATLAND и флага трехмерных точек функции INITGET. Как видим из таблицы, если системная переменная FLATLAND установлена в единицу, а флаг "трехмерные точки" функции INITGET не установлен, то функция getdist будет обрабатывать задаваемую трехмерную точку как двумерную, проецируя ее на текущую ПСК:

FLATLAND	*INITGET*	*Возвращается ли расстояние между трехмерными точками?*
0	1	Да
0	0	Да
1	1	Да
1	0	Нет

В Автокаде 11-й версии системная FLATLAND переменная всегда установлена в нуль. Поэтому управление функцией GETDIST происходит иначе. (См. описание функции INITGET.)

(getenv имя_переменной)

Возвращает строковое значение, присвоенное переменной среды DOS. Здесь имя переменной - строковая константа, представляющая собой имя переменной среды. Если эта переменная не определена, возвращается nil.

(getint [текст запроса-подсказки])

Ввод целого числа.

(getkword [текст запроса-подсказки])

Запрашивает у пользователя ключевое слово, определенное ранее при помощи функции INITGET. Если ответ не совпадает ни с одним определенным ключевым словом, Автокад попросит повторить ввод. Если ответ совпадает с ключевым словом, функция GETKWORD воз-

вращает это ключевое слово как строковую константу. Пустой ввод, если разрешен, возвращает nil.

Пример:

```
(initget 1 "Yes No")
(setq x (getkword "Are you sure? Yes/No "))
```

запросит пользователя и установит в символ X либо "Yes", либо "No" в зависимости от ответа. Можно отвечать только одной начальной буквой и неважно, строчной или прописной. В данной ситуации пустой ввод не разрешен, поэтому при неправильном ответе Автокад попросит повторить ввод.

(getorient [точка] [текст запроса-подсказки])

То же, что и GETANGLE, однако измерение угла осуществляется относительно текущего направления измерения углов, а не относительно нулевого направления оси X (абсолютный угол, в отличие от GETANGLE, которая измеряет относительный угол: угол относительно нулевого направления оси X).

(getpoint [точка] [текст запроса-подсказки])

Позволяет ввести точку. Если первый аргумент присутствует, Автокад рисует "резиновую" линию от точки, определяемой первым аргументом. Указание трехмерных точек обрабатывается так же, как и в функции GETDIST.

(getreal [текст запроса-подсказки])

Позволяет вводить действительное число. Ввод может быть осуществлен только с клавиатуры (или через планшет).

(getstring [флагпробела] [подсказка])

Запрашивает строковую константу и возвращает ее. Если строка длиннее 132 символов, то возвращаются первые 132 символа. Если в введенной строке имеются символы обратной косой черты, то они заменяются на две обратные косые черты.

Если флаг пробела присутствует и не равен nil, вводимая строка может содержать пробелы и завершением ввода считается нажатие клавиши "RETURN".

(getvar имяпеременной)

Возвращает значение системной переменной Автокада. Имя переменной должно быть заключено в кавычки (см. функцию SETVAR).

Как видим, функции семейства GET имеют много общего. Но кроме сходства их объединяет то, что все режимы ввода устанавливаются одной и той же функцией INITGET:

(initget [биты] [строка])

Данная функция устанавливает различные режимы, в которых работают все функции семейства GET, кроме функций GETSTRING и GETVAR. Возвращается всегда nil. Биты - необязательное целое число, биты которого интерпретируются следующим образом:

Бит INITGET	*Значение бита*
1	Запрещение пустого ввода
2	Запрещение ввода нуля
4	Запрещение ввода отрицательных чисел
8	Не контролируются лимиты, даже если включена системная переменная LIMCHECK
16	Возвращаются не двумерные, а трехмерные точки
32	"Резиновая" линия и рамка рисуются пунктиром
64	Игнорируется координата Z трехмерной точки (только в GETDIST Автокада 11 и 12)
128	Возвращается произвольный код с клавиатуры (для 12-й версии)

Биты задаются целыми значениями, являющимися степенями двойки, как показано в таблице, и общее целое число формируется путем сложения желаемых целых чисел. (Получается логическое И битов.) Следует избегать установки недокументированных битов, поскольку последние версии Автокада могут их использовать. Если установленные для ввода правила не выполнены, то Автокад попросит повторить ввод. Управляющие биты воспринимаются теми функциями, для которых они имеют смысл.

Второй аргумент функции INITGET, строка, используется для задания списка ключевых слов при вводе. Использование ключевых слов описано выше.

1.13. Логические функции Автолиспа

Автолисп предоставляет богатые возможности конструирования логических выражений и выполнения над ними логических операций.

Логический оператор - это функция, сравнивающая между собой два или больше аргумента. Результат сравнения (т. е. некоторое утверждение, касающееся двух аргументов) может быть либо истиной (T), либо ложью (nil). Основные логические функции - это И (AND), ИЛИ (OR) и НЕ (NOT):

Логическое выражение	*Результат*
(and выражение1 выражение2 ...)	Возвращает результат выполнения логического И над списком выражений. Возвращается nil, если любое (хотя бы одно) из выражений имеет значение nil, в противном случае - T
(not элемент)	Возвращает результат выполнения логического НЕ над своим аргументом
(or выражение1 выражение2 ...)	Возвращает результат выполнения логического ИЛИ над списком выражений. Возвращается nil, если все аргументы имеют значение nil; если при оценке аргументов слева направо встречается выражение не - nil, возвращается T
(= атом1 атом2 ...)	Возвращает T, если все атомы равны, в противном случае возвращается nil
(/= атом1 атом2)	Возвращает T, если атом1 не равен атому2. В противном случае возвращается nil. Функция не определена для числа аргументов большего двух
(< атом1 атом2 ...)	Возвращается T в том случае, если каждый последующий атом меньше предыдущего. В противном случае возвращается nil
(<= атом1 атом2 ...)	Возвращается T в том случае, если каждый последующий атом меньше или равен предыдущему. В противном случае возвращается nil
(> атом1 атом2 ...)	Возвращается T в том случае, если каждый последующий атом больше предыдущего. В противном случае возвращается nil
(>= атом1 атом2 ...)	Возвращается T в том случае, если каждый последующий атом больше или равен предыдущему. В противном случае возвращается nil
(eq выражение1	Определяет, идентичны ли выражение1 и выраже-

выражение2)	ние2, т. е. являются ли они фактически одним объектом или нет (т. е. порожден ли один из другого при помощи функции SETQ). EQ возвращает T, если оба выражения идентичны, в противном случае - nil. Обычно функция применяется для определения, являются ли два списка фактически одним или нет. Например, пусть (setq f1 '(a b c)) (setq f2 '(a b c)) (setq f3 f2) Тогда (eq f1 f3) возвращает nil (f1 и f3 - разные списки), а (eq f3 f2) возвращает T (f3 и f2 - один и тот же список) (См. также ниже функцию EQUAL.)
(equal выражение1 выражение2 [допуск])	Определяет, равны ли выражение1 и выражение2, т. е. равны ли их значения. Например, пусть (setq f1 '(a b c)) (setq f2 '(a b c)) (setq f3 f2) Тогда (equal f1 f3) возвращает T (значением f1 и f3 является одно и то же), а (equal f3 f2) возвращает T (f3 и f2 - в точности один и тот же список). Заметим, что, если оба списка EQUAL, они могут не быть EQ. Атомы, которые EQUAL, всегда к тому же EQ. Добавим, что два списка, которые EQ, всегда EQUAL. При сравнении двух действительных чисел (или двух списков действительных чисел, например точек) важно осознавать, что два "идентичных" числа могут незначительно отличаться друг от друга, если они вычислялись различными методами. Однако факультативный численный аргумент допуск позволяет задать максимальную точность, с которой могут отличаться друг от друга выражение1 и выражение2, оставаясь при этом EQUAL. Например, пусть (setq a 1.123456) (setq b 1.123457) Тогда (equal a b)возвращает nil (equal a b 0.000001) возвращает T

Кроме логических операций, производимых над выражениями как над целыми объектами, Автолисп включает в себя встроенные функции, производящие логические операции побитово над целыми числами:

Логическая операция	*Результат*
(~ целое)	Возвращает логическое НЕТ (дополнение до единицы) своего аргумента. Аргумент должен быть целым числом
(Boole кодфункции целое1 целое2 ...)	Определяет битовую функцию Булевой алгебры, производимой над двумя целыми аргументами. Функции кодируются следующим образом:

Целое1	*Целое2*	*Функция*
0	0	8
0	1	4
1	0	2
1	1	1

Некоторые значения этой функции эквивалентны стандартным булевым операциям:

Код функции	*Булева функция*
1	И
6	Исключающее ИЛИ
7	ИЛИ
8	НЕТ

(minusp элемент)	Функция возвращает T, если элемент является целым или действительным отрицательным числом. В противном случае возвращается nil
(zerop элемент)	Функция возвращает T, если элемент является целым или действительным нулем. Иначе возвращается nil
(numberp элемент)	Функция возвращает T, если элемент является целым или действительным числом. Иначе возвращается nil
(logand число число ...)	Функция возвращает результат действия побитового И над списком чисел. Числа должны быть целыми, и результат будет целым. Работает функция следующим образом. Каждое число представляется в двоичном виде; затем соответствующие разряды чисел собираются в список. Если в этом списке находится хотя бы один нуль, соответствующий разряд результирующего числа будет нулем; если в списке все единицы, разряд результирующего числа равен единице

(logior число число ...)	Функция возвращает результат действия побитового ИЛИ над списком чисел. Числа должны быть целыми, и результат будет тоже целым. В этой функции каждое число также представляется в двоичном виде; затем соответствующие разряды чисел собираются в список. Если в этом списке находится хотя бы одна единица, соответствующий разряд результирующего числа будет единицей; если в списке все нули, разряд результирующего числа равен нулю
(listp элемент)	Возвращает T, если элемент является списком, в противном случае - nil
(atom элемент)	Возвращает nil, если элемент является списком, в противном случае - T. Любой элемент, который не является списком, считается атомом
(null элемент)	Функция возвращает T, если элемент вычисляется в nil, противном случае - nil
(boundp атом)	Функция возвращает T, если атом имеет некоторое значение, в противном случае - nil

1.14. Условное ветвление программ

Каждая программа имеет свою логическую структуру. Ветвление - это способ управления ходом выполнения программы. Условные операторы - это средство управления ветвлением программ. Автолисп имеет две функции ветвления, IF и COND. Условное ветвление подразумевает проверку какого-то условия, по результатам которой осуществляется переход. Условия содержатся в условных выражениях, использующих операторы отношения и логические операторы. Условные выражения могут содержать любые выражения Автолиспа.

(cond (тест1 результат1 ...) ...)

Воспринимает в качестве аргументов любое число списков. Она оценивает по очереди первые элементы списков до тех пор, пока не встретится элемент, отличный от nil. Затем вычисляется то выражение, которое следует за тестом, и возвращается значение последнего выражения в субсписке. Если в субсписке только одно выражение (например, результат отсутствует), то возвращается значение выражения тест. COND - основная функция условия в Автолиспе. Как видно из следующего примера, в котором функция COND возвращает нуль или единицу

в зависимости от введенной пользователем строковой переменной s, эта функция может использоваться в качестве условного оператора переключения ("case" во многих языках программирования):

```
(cond     ((= s "Y") 1)
          ((= s "y") 1)
          ((= s "Д") 1)
          ((= s "д") 1)
          ((= s "N") 0)
          ((= s "n") 0)
          ((= s "Н") 0)
          ((= s "н") 0)
)
```

(if тест-выражение выражение-тогда выражение-иначе)

Эта функция использует выражение по условию. Если тест-выражение не nil, то исполняется выражение-тогда, в противном случае - выражение-иначе. Последнее выражение не обязательно. Возвращает значение выполненного по условию выражения; если выражение-иначе отсутствует и тест-выражение - nil, то IF возвращает nil.

Иногда требуется по условию выполнить не одно, а несколько выражений, в то время как функция IF не позволяет этого. В этом случае используют функцию PROGN, которая последовательно вычисляет каждое выражение и возвращает значение последнего: (progn выражение1 выражение2 ...). Например:

```
(if (= a b)(progn
        (setq a (+ a 10))
        (setq b (- b 10))
        )
)
```

1.15. Организация циклов

Как и многие языки программирования, Автолисп имеет средства организации повтора выполнения групп операторов. Циклы полезны, например:

- для уменьшения числа операторов в программе (в случае повтора однотипных действий);
- для повтора выполнения процедуры до отмены ее пользователем;
- для получения решения математической задачи со сходимостью;
- для пакетной обработки списков данных (например, при работе DXF-файлом).

Оператор организации циклов Автолиспа аналогичны командам циклов в других структурных языках. Следует отметить, однако, определенную бедность Автолиспа в этом отношении, обусловленную, вероятно, его спецификой.

Простейшим оператором повтора является оператор REPEAT. Эта функция повторяет любое число операторов указанное число раз. Возвращается последнее значение последнего выражения цикла. Синтаксис:

```
(repeat число выражение1 выражение2 ...)
```

Функция WHILE похожа на функцию REPEAT, однако число повтора не определено, а выход из цикла осуществляется по условию. В отличие от структуры IF оператор WHILE не включает в себя выполнение какого-либо выражения в том случае, если условие не выполняется. Как COND, REPEAT и PROGN, WHILE позволяет включать в свое тело неограниченное число операторов. В начале выполнения каждого цикла проверяется условие и, если оно выполняется, выполняется тело цикла, после чего опять проверяется условие, и так до тех пор, пока выражение-условие не станет ложным. Будьте внимательными: неаккуратное использование цикла WHILE может привести к зацикливанию программы. Цикл WHILE возвращает значение последнего вычисленного перед выходом из цикла выражения. Если выражение-условие изначально было ложным и вхождения в тело цикла не было, WHILE возвращает nil. Функцию WHILE удобно использовать для проверки правильности ввода, зацикливая ввод до тех пор, пока введенные данные не будут удовлетворять каким-то условиям (прежде чем использовать для этих целей функцию WHILE, убедитесь, что ту же задачу нельзя решить обычными средствами Автолиспа - getkword). Цикл WHILE можно использовать также для организации итераций. Итерация - это процесс повторения группы вычислений до тех пор, пока результат вычисления одного или более выражений не станет удовлетворять какому-то условию. Условное выражение итерационного цикла WHILE обычно содержит некоторую переменную, значение которой изменяется в процессе выполнения тела цикла. Синтаксис WHILE:

```
(while тест-выражение выражение1 выражение2 ...)
```

Еще одним типом функции - организатора цикла является функция FOREACH (буквально "для каждого"). Это специальное средство Автолиспа, предназначенное для работы со списками. Синтаксис:

```
(foreach имя список выражение1 выражение2 ...)
```

Эта функция, проходя по списку, присваивает каждому элементу имя и вычисляет каждое выражение для каждого элемента списка. Может быть задано неограниченное число выражений. Возвращается результат последнего вычисленного выражения. Например,

```
(foreach n '(a b c) (print n))
```

эквивалентно

```
(print a)
(print b)
(print c)
```

с той разницей, что FOREACH возвращает только значение последнего вычисленного выражения.

```
(apply функция список)
```

Данная функция выполняет функцию с аргументом, заданным списком.

Для выполнения какой-либо одной операции над всеми элементами одного или нескольких списков служит функция

```
(mapcar функция список1 список2 ...списокN)
```

Число списков должно соответствовать числу аргументов, требующихся для функции. Функции возвращают список результатов.

Если вам необходимо использовать свою функцию, она должна быть заранее определена с помощью DEFUN. Однако в Автолиспе есть механизм создания так называемой анонимной функции

```
(lambda аргументы выражение ....)
```

Она применяется вместе с MAPCAR и APPLY.

Примеры:

```
(apply '+ '(1 2 3))                    возвращает 6

(apply '(lambda (x y z)
          (* x (- y z)
          )
          (5 20 14)
)                                      возвращает 30

(mapcar '(lambda x y)
          (/ (+ x y) 2.0))
          '(1 2 3) '(5 6 7)
)                                      возвращает (3 4 5)
```

Самым значительным отличием языков семейства Лисп от других языков высокого уровня является возможность конструирования функций языка самой программой. Сначала вы можете составить выражение, а затем его выполнить. Для выполнения сконструированного выражения служит функция

```
(eval выражение)
```

Пример:

```
(setq a '(max))
(while (not (setq b (getreal "\Введите число: ")))
(setq a (cons b a)))
```

```
(setq a (reverse a)
(eval a)
```

В этом достаточно сложном примере последовательно вводятся числа, заносимые в список до тех пор, пока их ряд не закончится пустым вводом. В конце списка оказалось имя функции MAX. Затем этот список переворачивается так, чтобы имя функции оказалось на первом месте. Применение функции EVAL к полученному списку возвращает максимальное из введенных чисел.

1.16. Вызов команд Автокада из программы на Автолиспе

Ввод данных в программу на Автолиспе и вспомогательные вычисления над ними призваны служить автоматизации проектирования. Поэтому в Автолиспе не может не быть средства, позволяющего обращаться к командам Автокада из программ пользователя. Таким средством является функция Автолиспа COMMAND:

```
(command аргумент1 аргумент2 ...)
```

Эта функция выполняет команду Автокада из Автолиспа и всегда возвращает nil. Аргументы представляют собой команды Автокада и их опции; каждый аргумент вычисляется и посылается в Автокад как ответ на соответствующий запрос. Имена команд и опции представляются как строковые константы, двумерные и трехмерные точки - соответственно как списки из двух или трех действительных чисел. Пустая строка функции COMMAND равносильна нажатию пробела на клавиатуре. Вызов COMMAND без аргументов равносилен нажатию "CTRL C" с клавиатуры и прерывает большинство команд Автокада.

Если переменная Автокада CMDECHO установлена в нуль, то при выполнении этой функции на экран не будет выводиться выполнение команды Автокада.

На использование этой функции налагаются некоторые ограничения:

- функции семейства GET не могут быть вложены в функцию COMMAND. Следует присваивать все необходимые значения переменным заранее;
- с помощью функции COMMAND нельзя работать с такими командами Автокада, как ДТЕКСТ, ЭСКИЗ в 10, 11 и 12-й версиях и ЧЕРТИ, ПЕЧАТАЙ и ПАКЕТ в 10-й и 11-й версиях, а также с командами, объявленными пользователем (defun C:команда). В 12-й версии команда ПАКЕТ должна быть в программе последней;
- нельзя использовать восклицательный знак для указания команде Автокада значения переменной.

Если в строке аргументов команды, вызываемой функцией COMMAND, встречается ключевое слово "Pause", то функция COMMAND приостановит свое действие, чтобы пользователь непосредственно ввел значение (или произвел отслеживание). В этот момент можно выполнить "прозрачную" команду, после чего выполнение функции COMMAND возобновится. Это позволяет, в частности, в процессе выполнения функции COMMAND использовать команды 'ПОКАЖИ, 'ПАН и др. Пауза будет длиться до тех пор, пока не будет введен допустимый аргумент и пока не выполнятся все "прозрачные" команды. При этом, если функция COMMAND требует ввода, запрос может быть удовлетворен с помощью меню. Для приостановления действия меню следует указать обратную косую черту.

1.17. Специальные функции

Из Автолиспа можно управлять некоторыми режимами графического редактора не обращаясь к функциям Автокада. К встроенным функциям Автолиспа, управляющим графическим редактором, можно отнести следующие функции:

(graphscr)

Переключает экран из текстового режима в графический в системах с одним экраном (эквивалентна клавише переключения экрана - F1 на IBM PC).

(textscr)

Переключает экран из графического режима в текстовый в системах с одним экраном (эквивалентна клавише переключения экрана - F1 на IBM PC).

(textpage)

Аналогична функции TEXTSCR за исключением того, что она очищает текстовый экран от любого отображаемого на нем текста. Функция появилась в 11-й версии Автокада.

(grclear)

Очищает текущий видовой экран. На одноэкранных системах это первоначально приведет к переключению из текстового режима в графический; при этом зоны команд и состояния меню останутся без из-

менения. Первоначальное состояние графического экрана может быть восстановлено с помощью функции REDRAW без аргументов.

(redraw [имя_примитива [режим]])

Действие данной функции зависит от количества аргументов. Если аргументов нет, то она перерисовывает текущий видовой экран, как это делает команда Автокада 'REDRAW ('ОСВЕЖИ): если она вызывается с аргументом имя_примитива, то перерисован будет только этот выбранный примитив. Эту функцию часто используют для идентификации примитива на экране после использования функции Автокада GRCLEAR (имена примитивов описаны в разделе "Доступ к примитивам и средствам Автокада" настоящей главы). Полный контроль за перерисовкой примитива обеспечивается заданием параметра режим, который может принимать одно из следующих значений:

Режим REDRAW	*Действие*
1	Перерисовывает примитив на экране
2	Не рисует примитив (стирает)
3	Подсвечивает примитив (если позволяет дисплей)
4	Перестает подсвечивать примитив (если позволяет дисплей)

Если имя_примитива - заголовок сложного примитива (полилинии или блока с атрибутами), то в процессе перерисовки будут участвовать как основной примитив, так и все подпримитивы, при условии, что аргумент режим положителен. Если же аргумент режим отрицателен, то в процессе выполнения функции REDRAW будет участвовать только основной примитив. Функция REDRAW всегда возвращает nil.

(grdraw от к цвет [подсветка])

Данная функция рисует вектор между двумя точками на текущем видовом экране. Координаты точек от и к определяют конечные точки отрезка в текущей ПСК. Вектор подрезается до видимой части на экране. Цвет вектора задается целочисленным аргументом цвет. Если это значение равно -1, то цвет дополняет цвет линии, поверх которой он отрисовывается, до цвета фона, что делает данный фрагмент невидимым. Если аргумент подсветка присутствует и не равен нулю, то вектор изображается подсвеченным, как выбранный примитив, если дисплей в состоянии это отобразить. При отсутствии или при нулевом значении этого аргумента используется обычный режим отрисовки.

(grtext [бокс текст [подсветка])

Данная функция обеспечивает доступ к текстовым частям графического экрана Автокада. Если аргумент бокс присутствует и равен целому числу в пределах от нуля до наибольшего нумерованного бокса экранного меню минус единица, высвечивается строковый аргумент текст в заданном боксе экранного меню. Если текст не помещается, он усекается. При слишком коротком тексте он дополняется пробелами.

В присутствии подсветки не равной нулю текст подсвечивается. Снять подсветку можно установкой этого значения в нуль. Подсветка другого бокса автоматически снимает подсветку предыдущего. При вставке текста в бокс он сначала должен быть записан без подсветки, а потом его можно подсветить. Для того, чтобы подсветить бокс меню или снять с него подсветку, необходимо указать тот же строковый аргумент текст, который был первоначально записан в этот бокс, но с аргументом подсветка. Заметим, что эта функция только высвечиват указанную строку экранного меню; она не заменяет отмеченных пунктов меню. Более того, на некоторых типах мониторов действие обычной подсветки пункта меню заключается в изменении цвета этого пункта, так что GRTEXT может возвратить текст пункта в предыдущее состояние, если этот пункт меню подсветить. На некоторых мониторах зона меню переписывается при выполнении действий по переключению экрана или при перерисовке экрана командами ОСВЕЖИ, РЕГЕН. Однако на большинстве мониторов текст функции GRTEXT будет оставаться в зоне экранного меню до тех пор, пока его не перепишет новая страница меню.

Если в функции GRTEXT указать номер бокса равным -1, текст будет записан в статусную строку режимов. Длина статусной строки различна на разных мониторах, в большинстве случаев она составляет 40 символов. При превышении этой длины текст усекается.

Если номер бокса равен -2, текст записывается в строку координат. Если отслеживание координат включено, значения, записанные в этом поле, будут переписываться, как только от устройства указания последуют новые данные о координатах. В двух последних случаях при заданном аргументе подсветка он игнорируется.

Если функцию вызвать без аргументов, текстовые поля экрана придут к стандартному виду.

(grread [отслеживание])

Функция позволяет непосредственно считывать информацию с устройств ввода Автокада, отслеживая перемещения устройства указания. Аргумент отслеживание, если он присутствует и не равен nil, дает воз-

можность возвратить координаты от устройств указания в то время, как они движутся, не требуя нажатия кнопки ввода.

GRREAD возвращает список, первый элемент которого - код, указывающий тип устройства или режима ввода.

Коды для первого элемента списка:

Код	*Назначение*
2	Символ клавиатуры - второй элемент - его ASCII-код
3	Выбранная точка - координаты в виде списка
4	Выбранная ячейка экранного меню - номер бокса
5	Задан режим отслеживание, не равный nil. Второй элемент - координаты режима отслеживания
6	Выбран пункт меню BUTTONS - номер кнопки
7	Выбран пункт меню TABLET1 - номер бокса
8	Выбран пункт меню TABLET2 - номер бокса
9	Выбран пункт меню TABLET3 - номер бокса
10	Выбран пункт меню TABLET4 - номер бокса
11	Выбран пункт меню AUX1 - номер бокса
12	Координаты, связанные с кнопкой указателя в качестве второго элемента. Всегда следует за типом 6, возвращает список

Нажатие клавиши "CTRL C" при выполнении функции GRREAD приведет к прерыванию выполнения Лисп-программы. Любой другой ввод будет передаваться непосредственно к функции, предоставляя полный контроль над устройством ввода.

При нажатии кнопки выбора и указания области экранного или падающего меню GRREAD возвращает код 11, но при последующем вызове код 12 не возвращается: код 12 возвращается только после кодов 6 или 11, если кнопка выбора нажата при указании на графическую область экрана.

Важно очищать буфер от данных кода 12 перед выполнением других действий с кнопкой выбора или меню AUX1. Для этого можно выполнить вложенную функцию GRREAD:

```
(setq code12 (grread (setq code (grread))))
```

Данная последовательность запоминает значение кода 12 как потоковый ввод с устройства.

(vports)

Возвращает список дескрипторов действующих в настоящий момент видовых экранов. Каждый дескриптор видового экрана - это спи-

сок, содержащий номер видового экрана и координаты его нижнего левого и правого верхнего углов. Значения координат находятся в интервале от 0.0 до 1.0, где (0.0 0.0) - координаты нижнего левого угла зоны графического экрана, а (1.0 1.0) - координаты ее верхнего правого угла. Дескриптор текущего видового экрана всегда стоит в списке первым.

Таким образом, из программы на Автолиспе можно гибко управлять режимами отображения графического редактора. Но на этом не исчерпывается список ориентированных на систему Автокад функций Автолиспа. В Автолиспе есть функции, позволяющие использовать внутренние возможности графического редактора как системы управления графической базой данных. К таким функциям относятся функции объектной привязки и определения различных геометрических параметров объектов, описываемые ниже.

(menucmd строка)

Данная программа представляет механизм листания меню в программах на Лиспе, что позволяет согласованно работать с загруженными файлами меню, высвечивая соответствующие страницы. MENUCMD всегда возвращает nil. Аргумент строка записывается в следующей форме:

раздел=субменю

Раскроем, что имеется в виду под разделом и субменю.

Раздел	*Имя субменю*
S	SCREEN
B1 - B4	BUTTONS от 1 до 4
I	ICON
P1 - P16	(POP) от 1 до 16
T1 - T4	TABLET от 1 до 4
A1 - A4	AUX от 1 до 4
M	Выражения DIESEL

Субменю - Указывает, какое субменю активизировать. Именем должна быть либо одна из меток субменю без ** в текущем загруженном файле меню, либо имя одного из разделов меню. Знак S здесь не используется.

Что касается графических и падающих меню, то для них допустимо имя *, которое означает вызов субменю из текущего раздела.

1.18. Геометрические функции

В этом разделе собраны функции пересчета одних параметров, определяющих построение графических примитивов из Автолиспа, в другие.

(osnap точка режим)

Возвращает точку, которая является результатом применения объектной привязки, задаваемой в строке "режим" для указанной точки. Режим - строковая константа, состоящая из одного или более идентификатора объектной привязки, как, например, "середина", "центр" и т. д., разделенных запятыми:

```
(setq pt2 (osnap pt1 "сер"))
(setq pt2 (osnap pt1 "сер,кон,цен"))
```

Если аргумент точка - двумерная точка (список из двух действительных чисел), то будет возвращена двумерная точка. Если точка - трехмерная точка (список из трех действительных чисел), то будет возвращена трехмерная точка. Если ни одной точки, соответствующей заданному режиму объектной привязки, не найдено, то будет возвращен nil. Отметим, что действие этой функции зависит также от текущего трехмерного вида и от значения системной переменной FLATLAND.

(polar точка угол расстояние)

Возвращает точку в ПСК, находящуюся под заданным углом и расстоянием от заданной точки; угол измеряется в радианах в направлении против часовой стрелки от оси X. Хотя точки могут быть и трехмерными, угол всегда определяется в текущей плоскости построений. Если значение системной переменной FLATLAND равно нулю, то возвращаются трехмерные точки, в противном случае - двумерные.

(distance точка1 точка2)

Эта функция возвращает расстояние между двумя трехмерными точками. Если значение системной переменной FLATLAND не равно нулю, то функция DISTANCE предполагает двумерные точки (игнорирует координату Z переданной точки) и возвращает расстояние между точками - проекциями указанных трехмерных точек на текущую плоскость построений.

(angle точка1 точка2)

Возвращает угол в радианах, образованный лучом, направленным из точки1 в точку2 и осью X текущей плоскости построений. Угол измеряется против часовой стрелки. Трехмерные точки проецируются на текущую плоскость построений.

(inters точка1 точка2 точка3 точка4 [неопред])

Возвращает точку пересечения двух отрезков (точка1 точка2) и (точка3 точка4). Все точки выражены в координатах текущей ПСК. Если значение системной переменной FLATLAND равно нулю, то точки считаются трехмерными и контролируется пересечение в трехмерном пространстве. В противном случае отрезки проецируются на текущую плоскость построений и пересечение контролируется на плоскости. Если факультативный аргумент неопред присутствует и является nil, то контролируется пересечение не отрезков, а определяемых ими прямых и INTERS будет возвращать точку пересечения даже в том случае, если она не принадлежит ни одному из отрезков. Если же факультативный аргумент неопред отсутствует или не является nil, то точка пересечения должна принадлежать обоим отрезкам (отрезки должны пересекаться), иначе будет возвращен nil. В Автокаде 11 пересечение контролируется в трехмерном пространстве, если все четыре точки имеют три координаты.

(trans точка из в [вектор])

Эта функция преобразует координаты точки (или величину перемещения) из одной системы координат в другую. Аргумент точка - список из трех действительных чисел, который можно интерпретировать либо как трехмерную точку, либо как трехмерное перемещение (вектор); из - код системы координат, в которой находится указанная точка, а в - код системы координат, в которой происходит преобразование координат точки. Если имеется факультативный аргумент вектор и его значение не равно нулю, то аргумент точка будет трактоваться как трехмерное перемещение.

1.19. Работа с файлами: ввод/вывод

Работая с Автолиспом, у вас рано или поздно возникнет желание записать ту или иную информацию в файл или принять ее оттуда. Средства работы с файлами в Автолиспе не слишком развиты, однако для большинства повседневных нужд этого вполне достаточно.

(open имя_файла режим)

Эта функция открывает файл. Возвращается дескриптор файла.

Функции ввода/вывода работают не с именами файлов, а с так называемыми дескрипторами файлов, определяющими не только собственно файл, но и режим доступа и различные другие системные параметры. "Открыть файл" - значит подготовить дескриптор файла к использованию его функциями ввода/вывода Автолиспа. Поэтому возвращаемое функцией OPEN значение дескриптора файла должно присваиваться некоторой символьной переменной, например:

```
(setq a (open "file.ext" "r"))
```

Здесь переменная a - дескриптор файла file.ext, открытого для чтения. Флаг чтения или записи - это, как и имя файла, строковая константа, состоящая из одной буквы, которая должна быть набрана на нижнем регистре. Допустимые значения флага чтения/записи приводятся ниже:

Режим OPEN	*Описание*
r	Открыть файл для чтения. Если файл с указанным именем не существует, возвращается nil
w	Открыть файл для записи. Если файл с указанным именем не существует, создается и открывается новый файл. Если такой файл уже существует, то хранящиеся в нем данные будут утеряны

Имя файла может включать в себя путь (имена надкаталогов). В системах MS-DOS/PC-DOS допускается также использовать букву устройства ввода/вывода; и вы можете пользоваться обратной косой чертой вместо прямой косой черты. (Напомним: чтобы ввести в строку одну обратную косую черту, следует напечатать две обратных косых черты: "\\").

(close дескриптор_файла)

Эта функция завершает все процессы, инициированные функцией OPEN. После закрытия файла использование дескриптор_файла невозможно, хотя сам дескриптор файла не изменяется. Использование функции close обязательно: все открытые файлы должны быть закрыты, иначе данные могут быть потеряны.

(findfile имя_файла)

Эта функция отыскивает файл имя_файла (т. е. возвращает полное имя файла по указанному основному имени файла) в каталогах, хранящих файлы Автокада. Сначала просматривается текущий каталог, затем

каталог, в котором хранится текущий рисунок; и, наконец, каталог, имя которого записано в переменной среды ACAD. Такое полное имя можно передавать функции open.

(read строка)

Эта функция возвращает первый список или атом из данной строки, причем строка не должна содержать пробелов, за исключением, находящихся в списке. READ возвращает аргументы, задавая им соответствующий тип данных.

(read-char [дескриптор_файла])

Считывает единичный символ из буфера клавиатуры или из открытого файла, заданного аргументом дескрипторфайла. Возвращается целое число - код ASCII считанного символа. Если дескрипторфайла не задан и буфер клавиатуры пуст, функция ожидает ввода символа (пользователь должен ввести что-либо с клавиатуры, заканчивающееся RETURN). Например, если в ответ на запрос функции read-char пользователь введет с клавиатуры “ABC” и завершит ввод нажатием клавиши RETURN, то функция read-char возвратит код 65 (код ASCII латинской буквы “A”). При следующих трех обращениях к read-char она возвратит соответственно 66, 67 и 10 (код перевода строки). Если последует пятый вызов функции read-char, она снова будет ожидать ввода. Для унификации работы программ на Автолиспе в различных операционных системах функция read-char возвращает при чтении конца строки один символ с кодом 10.

(read-line [дескриптор_файла])

Данная функция считывает строку символов с клавиатуры или из открытого файла, заданного аргументом дескриптор_файла. Возвращается считываемая строка. Если достигнут конец файла, возвращается nil.

(write-char число [дескриптор_файла])

Эта функция записывает один символ на экран или в открытый файл, заданный аргументом дескриптор_файла. Здесь число - код ASCII символа и является возвращаемым функцией значением. Функция write-char не может записать в файл символ NUL (ASCII - код 0).

(write-line строка [дескриптор_файла])

Эта функция записывает строковую константу строка на экран или в открытый файл, заданный аргументом дескриптор_файла. Она возвращает строку, взятую в кавычки, и опускает кавычки при записи в файл.

(prin1 [выражение[дескриптор_файла]])

Данная функция выводит выражение на экран или в открытый файл, заданный аргументом дескриптор_файла, без пробелов и переходов на новую строку и возвращает выражение. Выражение может быть любым, а не только строковой константой. Если выражение - строковая константа, включающая управляющие символы, то они интерпретируются следующим образом:

Символ	*Значение*
\\\	символ \
\”	символ “
\e	символ escape-последовательности
\n	переход на новую строку
\r	возврат каретки
\t	табуляция
\nnn	символ с восьмеричным кодом nnn

При использовании PRIN1 без аргументов в качестве последнего выражения определяемой пользователем функции, то после завершения работы функции все, что появится на экране, - это пустая строка.

(princ [выражение [дескриптор_файла]])

Данная функция в отличие от предыдущей предназначена для печати управляющих символов в выражении без расширения, т. е. так, как они могли считываться функциями типа READ-LINE.

(print [выражение [дескриптор_файла]])

Данная функция в отличие от предыдущей предназначена для печати выражения с новой строки с последующим пробелом.

(terpri)

Данная функция печатает новую строку на экране и возвращает nil. Она не используется для ввода/вывода файлов.

(prompt сообщение)

Данная функция высвечивает сообщение в зоне подсказок экрана и возвращает nil. Сообщение - это строка символов.

1.20. Работа со списками

Как говорилось в п. 1.4.2, список - это группа элементов любого допустимого в Автолиспе типа. Список Автолиспа может содержать любое количество действительных и целых чисел, строк, переменных или других списков. Все, что находится между открывающей и закрывающей круглыми скобками, - список. Списки удобно использовать для организации и обработки больших массивов связанных данных. Автолисп имеет развитые средства работы со списками. Они вкратце описаны ниже.

(list выражение1 выражение2)

Эта функция просто составляет список из своих аргументов. Например, выражение (list 5.5 8.0) возвращает список, содержащий два действительных числа: (5.5S.0); этот способ часто используется в Автолиспе для создания новых точек из известных координат. Подчеркнем, что при составлении списка функция list оценивает выражения-элементы.

(quote выражение) или 'выражение

Функция quote (буквально - "цитировать") подавляет оценку своего выражения. Это бывает нужно при формировании списка. Например, выражение (list 'a 'b 'c) формирует список (a b c).

Поскольку список представляет собой группу элементов, может возникнуть необходимость извлечения одного элемента из списка. Существует две простейшие функции извлечения элементов списка:

(car список)

Возвращает первый элемент списка. Если список пуст, возвращается nil;

(cdr список)

Возвращает все элементы списка, кроме первого. Если список пуст, возвращается nil. Если элементом списка является точечная пара, функция возвращает последний элемент, не заключая его в список.

Из этих простейших функций, как из кирпичиков, составлены сцепления функций извлечения элементов списка вплоть до четырех уровней вложенности:

(caar x)	эквивалентно	(car (car x))
(cdar x)	эквивалентно	(cdr (car x))
(cadar x)	эквивалентно	(car (cdr (car x)))
(cadr x)	эквивалентно	(car (cdr x)
(cddr x)	эквивалентно	(cdr (cdr x))
(caddr x)	эквивалентно	(car (cdr (cdr x)))

В Автолиспе CADR часто используется для "извлечения" координаты Y двумерной или трехмерной точки, а CADDR - для извлечения координаты Z. Для этой же цели можно воспользоваться функцией nth:

(nth номер список)

Извлекает элемент списка с нужным номером.

Однако будьте осторожны при работе с этой функцией: первый элемент списка имеет номер 0, а не 1 (функция nth считает не 1, 2, 3,..., а 0, 1, 2, 3, ...). Так считает машина, однако вам-то от этого не легче - учтите, что в счете элементов списков у функций Автолиспа нет единообразия, поэтому нужно быть внимательным. Кроме того, к средствам поддержки списков относятся функции:

(last список)

Возвращает последний элемент списка, причем список не должен быть равен nil. НЕ РЕКОМЕНДУЕТСЯ использовать эту функцию для извлечения Y- или Z-координаты точки, поскольку это требует дополнительных усилий по поддержке правильной размерности точек;

(reverse список)

Возвращает список с элементами, переставленными в обратном порядке;

(length список)

Возвращает целое число, равное числу элементов в списке;

(append выражение1 выражение2 ...)

Берет любое число выражений (списков) и сливает их в один список;

(cons выражение список);

Эта функция, наряду с функцией list, используется для создания списков. Она добавляет в начало списка новый элемент, которым может быть также и выражение.

Кроме этих простейших функций существуют функции assoc и subst, позволяющие работать со структурированными списками, подобно структурам в других языках программирования. Работа со структурированными списками будет рассматриваться ниже, при разборе работы с графической базой данных Автокада (п. 7.22).

1.21. Создание функции в Автолиспе

Вы можете определять в Автолиспе свои собственные функции. Функция DEFUN определяет функцию посредством создания структурированного списка операторов программы. Определенная вами функция создает свою собственную замкнутую область локальных переменных. При вызове функции в эту замкнутую область передаются данные, выполняются операторы вашей программы, после чего осуществляется передача данных обратно в среду Автолисп - Автокад.

Формат функции DEFUN следующий:

```
(defun имя (аргументы / локальные параметры)
        тело функции
)
```

Правила именования функций те же, что и правила именования переменных. Помните, что если вы переопределили встроенную функцию Автолиспа, то она останется для недоступной до тех пор, пока вы не начнете новый рисунок. Строки программы составляют ядро функции. Порядок оценки их вычислителем Автолиспа уже обсуждался. Последнее возвращаемое значение передается в среду Автокад - Автолисп.

При запуске графического редактора Автокад загружает файл acad.lsp (если он существует) автоматически. В этот файл можно внести определения наиболее часто используемых функций, и они будут загружаться автоматически при вызове графического редактора. При этом на экране появится сообщение

```
Loading acad.lsp...loaded.
Загружаю acad.lsp...загружен.
```

При определении функции при помощи DEFUN можно указать, что она должна выполняться автоматически сразу же после загрузки. Для этого ее нужно назвать S::STARTUP. Если вы поместите определенную таким образом функцию в файл acad.lsp, то она будет автоматически загружаться и выполняться сразу же после вызова Автокада.

Определенные вами функции могут быть использованы так же, как и команды Автокада. Это позволяет сделать программу модульной, многократно использующей какие-то подпрограммы.

Для того, чтобы пользоваться вновь созданными командами Автокада, функции, определенные с помощью DEFUN, должны удовлетворять следующим условиям:

- Имя функции должно быть вида C:XXX. Причем, если вы работаете в русском Автокаде, все буквы должны быть прописными. Буква "C" набирается в латинском регистре. Часть имени "C:" должна присутствовать всегда, а часть имени "XXX" может быть любой, но не должна дублировать какую-либо команду Автокада, внешнюю или встроенную функцию.
- Функция должна быть определена без аргументов. Допускаются только локальные символы.

1.22. Установка текстового редактора

Между тем программу Автолиспа можно записывать в файл, что позволяет не набирать ее каждый раз заново, а загружать из файла (и конечно, редактировать). Файлы программ на Автолиспе имеют расширение .lsp. Программа может содержать любое количество определений функций и других выражений.

Файлы программ на Автолиспе являются обычными текстовыми файлами, и вам не надо беспокоиться о разбиении файла на страницы или считать пробелы. Пустые строки не рассматриваются интерпретатором Автолиспа как нажатия клавиши "ENTER", а просто игнорируются, как и лишние пробелы. Вы набираете программу-функцию в любом текстовом редакторе, не вставляющем в файл своих служебных символов (это может быть, например, строковый редактор EDLIN, входящий в комплект DOS, экранные редакторы Norton Editor, MultiEditor или даже Лексикон). Не рекомендуется для создания программы на Автолиспе использовать текстовые процессоры, такие, как Microsoft Word или CharWriter, поскольку они применяют свой формат файлов, вставляя в них свою служебную информацию, которая не может быть обработана компилятором Автолиспа (конечно, они позволяют выводить набранный документ в так называемом режиме ASCII без форматирования, но их назначение не набор программ, а создание документов).

В стандартной поставке Автокада в подменю UTILITY/ExternalCommands имеется пункт EDIT:, выбор которого приводит к автоматическому вызову строкового редактора EDLIN. При выборе EDIT на экране высвечивается подсказка:

File to edit:<Введите имя файла>

Редактировать файл:<Введите имя файла>

После того как вы введете имя файла, Автокад вызывает текстовый редактор. Если вы не настраивали ваш Автокад, то этот редактор - строковый редактор EDLIN. Если вы сталкивались с ним раньше, то знаете, что это не самый удобный редактор. Мы советуем подключить к Автокаду редактор Norton Editor. Его преимущество в том, что он невелик, т. е. требует мало памяти, и в то же время это достаточно мощный для нас экранный редактор, отслеживающий пары скобок и кавычек.

Все внешние команды Автокада определены в файле acad.pgp:

```
CATALOG,DIR /W,30000,*Files: ,0
DEL,DEL,30000,File to delete: ,0
DIR,DIR,30000,File specification: ,0
EDIT,EDLIN,42000,File to edit: ,0
SH,,30000,*DOS Command: ,0
SHELL,,127000,*DOS Command: ,0
TYPE,TYPE,30000,File to list: ,0

КАТАЛОГ,DIR /W,27000,*Показать файлы: ,0
DEL,DEL,27000,Удалить файл: ,0
DIR,DIR,27000,Показать файлы: ,0
РЕДАКТ,edlin,30000,Редактировать файл: ,0
ДОС1,,27000,*Команда ДОС: ,0
ДОС,,127000,*Команда ДОС: ,0
TYPE,TYPE,27000,Файл для просмотра: ,0
```

Каждая строка в этом файле соответствует одной команде подменю External Commands (Внешние команды). Это подменю называется так именно потому, что его команды берутся из файла acad.pgp, а не записаны в самом Автокаде. Можно модифицировать команды этого подменю. Рассмотрим одну строку (команду) этого файла, EDIT.

Первым элементом строки, выделенным запятой, является команда, высвечиваемая в Автокаде в подменю External commands. Вторым элементом является команда DOS, которая будет выполнена при выборе из меню команды. Третьим элементом командной строки является объем памяти, который освобождает Автокад перед вызовом редактора. Четвертым элементом строки является подсказка, которая высвечивается после выбора команды и требует ввести имя файла, который вы хотите редактировать. И наконец, пятым элементом строки является целое число, определяющее, какой экран (графический или текстовый) будет вызван при возврате в Автокад (0 - текстовый экран, 4 - графический). Для того чтобы вместо редактора EDLIN вызывался редактор Norton Editor, нужно заменить строку EDIT на следующую:

```
EDIT,ne,60000,File to edit: ,0

РЕДАКТ,ne,60000,Редактировать файл: ,0
```

Теперь файл с программой на Автолиспе редактировать удобнее.

1.23. Написание программ на Автолиспе

Вы уже знаете, что в Автолиспе углы измеряются в радианах и могут принимать значения от 0 до 2*PI радиан. Поскольку более удобно измерять углы в градусах, введем новую функцию, преобразующую градусы в радианы. Итак, войдите в редактор, указав ему имя myprog.lsp, и наберите текст следующей программы:

```
;Преобразование углов из градусов в радианы
(defun dtr (a)
        (* pi (/ a 180.0))
)
(prompt "\nПрограмма загружена - синтаксических ошибок нет!")
;Конец программы
```

Заметим, что строка в тексте программы, начинающаяся с точки с запятой, - это строка комментариев, которая Автолиспом не обрабатывается. Если точка с запятой стоит не в начале строки, то все, что после нее,- тоже комментарии. Расположение скобок не играет никакой роли при интерпретации программы вычислителем Автолиспа, но программа лучше читается, если закрывающие скобки располагаются либо на той же строке, либо в том же столбце, что и открывающие.

Сохраните файл с программой на диске и загрузите его в Автокад:

```
Команда:(load "myprog")
Автолисп возвращает: nil
```

Здесь load - функция Автолиспа, выполняемая с командной строки Автокада. Функция вызывает программу из файла в среду Автокад - Автолисп. Указывая ей имя файла, не надо писать расширение: она берет файлы с расширением .lsp по умолчанию. (Не путайте функцию load Автолиспа с командой LOAD Автокада, загружающей файлы форм, - это предупреждение относится особенно к тем, кто работает на английской версии Автокада.) При загрузке функции Автолисп читает из файла определение функции, осуществляет синтаксическую проверку и, если ошибок (синтаксических!) нет, сохраняет функцию в области памяти, специально отведенной для пользовательских программ, после чего отображает на экране имя загруженной функции. Все, что не является определением функции (в нашем случае это подсказка "prompt"), выполняется Автолиспом немедленно. Определенная пользователем функция должна быть вызвана явно.

Теперь можно проверить работу этой функции, вводя разные значения аргументов:

```
Команда:(dtr 180)
Автолисп возвращает: 3.141593
```

Аргументы функции - это имена переменных, которые используются для передачи данных в локальную среду функции. Аргументы функции называются также формальными параметрами (в отличие

от фактических параметров, т. е. тех параметров, которые указываются при вызове функции). В примере а - формальный параметр, число 180 - фактический. Список фактических параметров должен соответствовать списку формальных параметров, т. е. они должны иметь одинаковое число элементов, без пропусков и перестановок.

1.24. Доступ к примитивам и средствам Автокада

Чуть ли не самой главной особенностью Автолиспа является то, что он позволяет осуществлять доступ к графической базе данных (ГБД) Автокада, многократно умножая возможности адаптации последнего к какому-либо типу задач. Попытаемся вкратце показать, как можно использовать возможности Автолиспа для работы непосредственно с объектами чертежа.

Любой создаваемый в Автокаде чертеж состоит из отдельных примитивов, геометрическое описание которых хранится в специальном формате (формате Автокада) в файле чертежа (расширение .dwg). При загрузке чертежа Автокад заполняет графическую базу данных - заносит в нее системные настройки, создает список объектов и вносит в ГБД геометрическое описание этих объектов, присваивая при этом каждому примитиву уникальное имя. Итак, в сеансе редактирования каждый примитив Автокада (отрезок, дуга, окружность и т. п.) имеет свое имя, по которому его распознает сам Автокад. Поскольку имена меняются от одного сеанса редактирования к другому, не имеет смысла их хранить - даже не пытайтесь запомнить их. Вместо этого следует в программе на Автолиспе сначала найти имя примитива в базе данных Автокада с тем, чтобы впоследствии непосредственно манипулировать геометрическими характеристиками примитива или использовать их в макроопределениях меню. Попробуем извлечь это имя из ГБД при помощи Автолиспа. Нарисуйте отрезок и введите с командной строки Автокада строку

```
Command:(setq ename (entlast))
Автокад возвращает:<Entity name: 60000018>

Команда:(setq ename (entlast))
Автокад возвращает:<Имя примитива: 60000018>
```

Тем самым мы присвоили переменной ename имя последнего примитива (в данном случае отрезка). Напомним еще раз, что имена примитивов меняются от сеанса к сеансу, и вы наверняка увидите на экране другое имя. Имена примитивов в Автокаде - шестнадцатеричные величины; имя примитива может быть, например, таким: 60000A14. Используя это имя, вы можете при помощи функции ENTGET получить доступ к данным, связанным с примитивом:

```
(setq edata (entget ename))
```

Имя примитива - это новый для нас тип данных Автолиспа, и если функции Автолиспа ENTGET требуется имя примитива, то бесполезно указывать число 60000018 - надо передать переменную ename, в которой это имя хранится.

В результате выполнения команды вы получите малопонятное сообщение:

```
((-1 . <Имя примитива: 60000020>) (0 . "LINE") (8 . "0")
(10 1.0 2.0 0.0) (11 6.0 6.0 0.0))
```

Дело в том, что с точки зрения Автолиспа все данные, описывающие примитив, представляют собой список, состоящий, в свою очередь, из подсписков, в которых сгруппированы по функциональному назначению все данные о примитиве, как геометрические, так и общие: слой, цвет и т. п. Подсписки отличаются друг от друга по специальным кодам формата DXF (Drawing eXchange Format - формат обмена рисунками), которые позволяют определить, какой тип данных хранится в подсписке. Каждый подсписок имеет две части. Первая - код DXF, вторая - данные. Целое число 0, например, представляет собой код типа примитива. Код 8 говорит о том, что следующее за ним число - номер слоя. Код 10 - начальная точка примитива, код 11 - конечная, и т. п. Отметим, что набор кодов DXF различен для примитивов разных типов. Однако сами коды относятся ко всем примитивам - имя примитива, например, всегда хранится в подсписке с кодом DXF -1.

Представим полученный список edata в более понятном виде:

```
(       (-1 . <Имя примитива: 60000020>)
        (0 . "LINE")                      ; Тип примитива
        (8 . "0")                         ; Слой
        (10 1.0 2.0 0.0)                  ; Начальная точка
        (11 6.0 6.0 0.0)                  ; Конечная точка
)
```

Как нетрудно догадаться, пользуясь кодами DXF, можно извлечь из списка edata любую информацию о примитиве. Такой доступ к рисунку более сложен, но предоставляет невиданные ранее возможности. Теперь вы сможете изменять практически все свойства примитивов.

В Автокаде имеется стандартное средство работы не с одним, а с несколькими примитивами. Это средство называется набором. Практически все команды редактирования работают не с отдельными примитивами, а с группой примитивов. Из Автолиспа также можно работать с наборами примитивов - предоставлять пользователю возможность заносить примитивы в набор и затем модифицировать занесенные туда примитивы. Набор формируется функцией SSGET:

```
(ssget [режим] [точка1] [точка2])
```

Необязательный аргумент режим - это строка, которая указывает способ выбора примитива. Им может быть "Р", "С", "П" и "Т" - соответствующие Рамке, Секущей рамке, Последнему и Текущему набору

примитивов Автокада. Для 12-й версии добавляются "РМН", "СМН", "Л", соответствующие Многоугольнику, Секущему многоугольнику и Линии. Аргументы точка1 и точка2 - списки точек, указывающие точки, относящиеся к выбору.

Примеры функции SSGET:

```
(ssget)                       ; Выбирает по одному объекты чертежа
(ssget "Т")                   ; Выбирает текущий набор
(ssget "П")                   ; Выбирает последний примитив
(ssget '(2 2))                ; Выбирает примитив, проходящий через точку (2, 2)
(ssget "Р" '(0 0) '(5 5))     ; Выбирает примитивы в рамке от (0,0) до (5,5)
(ssget "С" '(0 0) '(4 5))     ; Выбирает примитивы, пересекаемые
                              ; рамкой от (0,0) до (4,5)
(ssget "X" список-фильтр)     ; Выбирает примитивы
                              ; в соответствии со списком-фильтром
```

Выбранные объекты подсвечиваются только тогда, когда SSGET используется без аргументов.

Особый режим - режим SSGET "X" - фильтры выбора. В этом режиме функция SSGET просматривает весь рисунок и создает набор, состоящий из имен всех основных примитивов, удовлетворяющих заданным критериям.

Доступные коды для SSGET "X" представлены в следующей таблице:

Код	*Значение*
0	Тип примитива
2	Имя блока для описания блока (INSERT)
6	Имя типа линии
7	Имя гарнитуры шрифта для определений текста или атрибутов
8	Имя слоя
38	Уровень (вещественное число)
39	Высота (вещественное число)
62	Код цвета (0="ПОБЛОКУ", 256= "ПОСЛОЮ")
66	Следующий за атрибутом флаг в описании блока (INSERT)
210	Вектор направления выдавливания (список из трех вещественных чисел)

Для первых пяти кодов в Автокаде 11 и 12 разрешается использовать глобальные символы.

Например, используя этот режим, можно создать набор, состоящий из всех отрезков, находящихся на слое "ЭТАЖ" зеленого цвета:

```
(ssget "x" '((0 . "LINE")(8 . "ЭТАЖ")(62 . 3)))
```

В Автолиспе имеются хорошие средства работы с наборами примитивов. Рассмотрим их вкратце.

Примитив	*Действие*
(sslength набор)	Возвращает число примитивов в наборе. Если это число больше 32 767, то оно возвращается как действительное
(ssname набор номер)	Возвращает имя примитива под номером из набора. Запомните, что первый примитив в наборе имеет номер 0! Если примитивов слишком много, используйте действительные числа (округляются до целых)
(ssadd имя_примитива набор)	Если вызвать функцию без аргументов, ssadd создаст новый набор без единого примитива. Если вызвать только с аргументом имя_примитива, будет создан набор, содержащий один примитив. Если вызвать с обоими указанными аргументами, примитив имя_примитива будет добавлен в набор
(ssdel имя_примитива набор)	Удаляет имя_примитива из набора и возвращает новый набор
(ssmemb имя_примитива набор)	Если примитив имя_примитива находится в наборе, то возвращается имя примитива, если нет - nil

Рассмотрим далее функции, позволяющие извлекать имена примитивов из ГБД по их порядку.

Функция	*Назначение*
(entnext [имя_примитива])	Будучи вызвана без параметров, возвращает имя первого неудаленного примитива в ГБД. Если задан аргумент имя_примитива, будет возвращено имя первого неудаленного примитива, следующего в ГБД за примитивом имя_примитива. Если примитива нет, будет возвращен nil. Возвращаются как основные примитивы, так и субпримитивы (например, вершины полилинии, полученные в результате аппроксимации последней сплайном)
(entlast)	Возвращается имя последнего основного неудаленного примитива в ГБД

Из Автолиспа можно непосредственно модифицировать данные о существующих в ГБД примитивах. Однако если мы хотим добавить

новый примитив, то должны использовать команды отрисовки или редактирования Автокада. Это ограничение связано с желанием защитить ГБД от неграмотного программиста: неправильно пользуясь командами Автокада, вы не сможете испортить саму ГБД - самое большее, что вы испортите, это свой чертеж.

Примитивы можно удалять из чертежа командой

```
(entdel ename)
```

Автолисп возвращает имя примитива, например:

```
<Entityname:60000018>
<Имя примитива: 60000018>
```

Интересно, что если вы повторите эту операцию, то примитив будет восстановлен. Этот способ позволяет удалять из ГБД невидимые примитивы, что невозможно обычным выбором объектов (нельзя выбрать невидимый объект).

Для того чтобы модифицировать геометрические характеристики примитива непосредственно в ГБД , надо уметь находить в DXF-списке данных примитива (наша переменная edata) подсписки, в которых хранится нужная информация. Выбор и изменение различных данных, относящихся к примитиву, осуществляется по коду DXF (это всегда целое число) с помощью функций assoc и subst:

(assoc элемент_списка список)

Извлекает из списка элемент списка по ключу элемент_списка. Если элемент_списка не найден, assoc возвращает nil.

При помощи этой функции можно извлечь, например, из списка goods ((1 "car" "volvo")(2 "price" 80000)) подсписок (2 "price" 80000):

```
(assoc 2 goods) возвращает список (2 "price" 80000)
```

(subst новый_элемент старый_элемент список)

Возвращает копию исходного списка с заменой всех найденных подсписков, идентичных старому_элементу, на новый_элемент. Если вхождений не обнаружено, subst возвращает копию старого списка (не nil!):

```
(subst '(2 "price" 100000) '(2 "price" 80000) goods)
```

возвращает:

```
((1 "car" "volvo")(2 "price" 100000))
```

Используя эту технику, попробуем извлечь из списка edata имя примитива:

```
Команда:(assoc 0 edata)
Автолисп возвращает:(0 . "LINE")
```

В приведенном только что примере мы фактически сказали Автолиспу: "Возврати мне подсписок с DXF-кодом 0". Автолисп просмотрел DXF-список примитива, нашел подсписок с кодом 0 и возвратил его. Разумеется, целиком. Полученный по коду подсписок все еще содержит DXF-код, который необходимо убрать: первый элемент списка, уже "отработал" свое и больше не понадобится. Для этой цели лучше всего использовать функцию CDR:

```
Команда:(cdr (assoc 0 edata))
Автолисп возвращает:"LINE"
```

Извлекая из DXF-списков нужную информацию, можно программно обрабатывать ее и затем, внеся изменения в DXF-список примитива при помощи функции subst, модифицировать ГБД при помощи функций entmod и entupd.

(entmod список)

С помощью данной функции преобразуется список в формат, возвращаемый функцией ENTGET, и обновляется информация базы данных о примитиве, имя которого указано в группе -1 списка. С помощью функции ENTMOD нельзя изменить тип примитива. Все объекты, на которые ссылается список примитива (гарнитура шрифта, тип линии, имена форм и блоков), должны быть известны Автокаду к моменту ее вызова. Исключением является имя слоя: если неизвестный Автокаду слой поименован в списке, ENTMOD создает новый слой со стандартными значениями.

(entupd имя_примитива)

Функция регенерации отдельного примитива, не обязательно основного.

Приведем пример того, как можно из Автолиспа модифицировать вершину полилинии:

```
(setq e1 (entnext))         ; Переменной e1 присваивается имя полилинии
(setq e2 (entnext e1))      ; Переменной e2 присваивается имя
                            ; первой вершины
(setq ed (entget e2))       ; В списке ed данные о вершине 1
(setq ed                    ; Модификация переменной ed
  (subst(10 1.0 3.7)        ; Этот список  помещается на место старого
  (assoc 10 ed)             ; вхождения списка с первым элементом 10
  )
)
(entmod ed)                 ; Обновляет данные в ГБД
(entupd e1)                 ; Обновляет полилинию на экране
```

Заметим, что если вы попытаетесь извлечь данные о вставке блока (INSERT), то получите информацию о самой вставке, а не об определении блока.

Итак, для модификации примитива (группы примитивов) непосредственно в базе данных нужно:

- предложить пользователю выбрать примитив (функция entsel) или группу примитивов (функция ssget). Используя функцию ssget, можно выбирать примитивы из ГБД программно, без обращения к пользователю - по их свойствам (тип примитива, слой и т. п.);
- извлечь из ГБД данные, относящиеся к этому примитиву (если использовался набор, то сначала следует извлечь имя примитива из набора - функция ssname, затем извлечь из ГБД данные, относящиеся к этому примитиву, - функция entget);
- средствами работы со списками Автолиспа модифицировать данные о примитиве в переменной - списке данных (функции assoc, subst);
- модифицировать саму ГБД: внести изменения в чертеж (функции entmod и entupd).

Информацию о слоях, типах линий, видах, гарнитурах шрифтов, блоках, ПСК, размерных стилях и видовых экранах Автокад хранит в справочных таблицах. Для работы с ними в Автокаде предусмотрены функции TBLNEXT и TBLSEARCH, которые просматривают таблицу и выдают содержащуюся в ней информацию. Содержимое таблиц изменять не разрешается.

(tblnext имятаблицы [первый])

Первым аргументом функции является строковая константа имени символьной таблицы. Допустимые имена таблиц - "LAYER", "LTYPE", "VIEW", "STYLE", "BLOCK", "UCS", "DIMSTYLE", "VPORT".

При повторном вхождении TBLNEXT возвращает следующее вхождение в указанную таблицу. Если аргумент первый присутствует и не равен nil, чтение таблицы начинается сначала. Если вхождений в таблицу нет, возвращается nil. Если вхождение найдено, возвращается список кодов и значений типа DXF. Точки входа, вызываемые из таблицы "BLOCK", содержат группу -2 с именем первого элемента в описании блока. Это имя можно использовать только в функциях ENTGET и ENTNEXT, т. е. они программно не модифицируются:

(tblsearch имятаблицы символ [следующий])

Если аргумент следующий присутствует и не равен nil, точка входа TBLNEXT изменяется так, чтовы следующий вызов TBLNEXT возвращал точку входа, следующую за возвращенной.

Пример:

```
(tblnext "BLOCK")
```

Возможный возврат:

```
(       (0 . "BLOCK")                        ; тип символа
        (2 . "BORDER")                       ; имя символа
        (70 . 0)                             ; флаги
        (10 . 9.7 2.0 0.0)                   ; X, Y, Z базовой точки
        (-2 . <Имя примитива: 40000126>)     ; первый примитив
)
```

Как уже говорилось, имена примитивов действительны только в текущем сеансе редактирования. Однако в Автолиспе есть средство, позволяющее узнавать примитивы из других сеансов редактирования. Это средство называется меткой. Метки - это просто номер примитива в ГБД. Метки присваиваются примитивам автоматически в процессе создания (если были включены), и с ними можно работать из Автолиспа. Для этого в Автолиспе существует функция handent.

(handent метка)

Возвращает имя примитива, который указывается строковым параметром метка. После того как получено имя примитива, оно может быть использовано для дальнейшей работы.

1.25. Работа с Лисп-программами

Для загрузки в память Автокада, как мы уже видели, используется команда (LOAD имя_файла [ошибка]), записываемая в командной строке. Ввод расширения .lsp не требуется.

Примеры разработки некоторых программ

В этой главе собраны некоторые примеры, с помощью которых мы попытаемся продемонстрировать приемы программирования отдельных задач, решаемых на Автолиспе.

2.1. Определение границ участка

Покажем, как можно определить границы участка поверхности, где расположено множество реперных точек-блоков, содержащихся в наборе. Определим функцию BOUND, формирующую минимальные и максимальные значения координат X и Y прямоугольника, куда входят все точки вставок блоков нашего чертежа.

Отключим объектную привязку и установим исходные значения требуемых переменных. Создадим набор NABP, куда входят все блоки чертежа:

```
(defun c:bound ()
(setq oss (getvar "osmode"))
(setvar "osmode" 0)
(setq nabfi nil nabtet nil nabfin nil
        nabtetn nil nabz nil n 0)
(setq nabp (ssget "x" '((0 . "INSERT"))))
```

Длина набора NABP - LN:

```
(setq ln (sslength nabp))
```

В цикле, выполняемом LN раз, сформируем два набора, содержащих все компоненты X и Y координат точек вставки блоков - NABX и NABY:

```
(repeat ln
 (setq pt (cdr (assoc 10 (entget (ssname nabp n)))))
 (setq nabx (cons (car pt) nabx))
 (setq naby (cons (cadr pt) naby))
 (setq n (1+ n))
)
```

Вычисление минимальных и максимальных значений координат теперь выполняется не просто, а очень просто:

```
(setq xmin (eval (cons 'min nabx)))
(setq xmax (eval (cons 'max nabx)))
(setq ymin (eval (cons 'min naby)))
(setq ymax (eval (cons 'min naby)))
```

Однако, если набор имеет очень много членов, может возникнуть ошибка: "Слишком много аргументов". Проверьте свою версию Автолиспа с помощью задания заданного числа блоков.

2.2. Развертка поверхности

Еще один пример - построение развертки трехмерной поверхности. Задачи подобного типа встречаются во многих приложениях. Раскрой заготовок является предварительным этапом изготовления швейных изделий, сложнопрофильных трубопроводов, изделий их жести и т. п.

С другой стороны, конструктор сначала проектирует общий вид изделия, а потом формирует развертку заготовки. Хотелось бы хотя бы частично автоматизировать этот процесс.

Займемся сначала простыми поверхностями типа цилиндрических и конических. Можно предложить следующий ход решения. Поскольку любые поверхности в Автокаде аппроксимируются трехмерными гранями, причем грани определены четырьмя угловыми точками и в некоторых ситуациях эти точки лежат не в одной плоскости, будем считать для простоты рассуждений, что грани, образующие поверхность, плоские. Выделим первую грань и повернем грань, соприкасающуюся с первой, так, чтобы они оказались в одной плоскости. Затем перейдем ко второй грани и проделаем ту же операцию. Ограничения, которые были приняты при постановке задачи, по сути означают, что каждая грань имеет только одного соседа.

Решение задачи в этой постановке выглядит следующим образом.

Определим функцию, решающую задачу:

```
(defun c:rslw ()
```

Перейдем в мировую систему координат:

```
(command "ПСК" "М")
```

Создадим набор под именем A, содержащий набор граней, подлежащих обработке, обозначив его длину LA. Кстати, если вы работаете на Автокаде-10 русифицированной версии и его серийный номер начинается с 30, функция ssget "x" записывается в виде ssget "Ш"!

```
(setq a (ssget "x" '((0 . "3DFACE")))
    la (sslength a))
```

Далее начинается цикл по всем граням набора:

```
(repeat (1- la)
```

Присваиваем очередной грани имя n3fdf:

```
(setq nf3df (ssname a 0)
```

Формируем список данных грани nf3df под именем f3df:

```
f3df (entget nf3df)
```

Дублируем этот список в aa:

```
aa f3df
```

Выделяем в виде списков координаты каждой из четырех точек грани nf3df в списках pr1, pr2, pr3, pr4:

```
pr1 (cdr (assoc 10 (cdr (cdr f3df))))
pr2 (cdr (assoc 11 (cdr (cdr f3df))))
pr3 (cdr (assoc 12 (cdr (cdr f3df))))
pr4 (cdr (assoc 13 (cdr (cdr f3df))))
```

Удаляем из набора a грань nf3df:

```
a (ssdel nf3df a)
```

и следующей грани присваиваем имя nf3df:

```
nf3df (ssname a 0)
```

а список ее данных именуем f3df:

```
f3df (entget nf3df)
```

со списками координат точек pr11, pr12, pr13, pr14:

```
pr11 (cdr (assoc 10 (cdr (cdr f3df))))
pr12 (cdr (assoc 11 (cdr (cdr f3df))))
pr13 (cdr (assoc 12 (cdr (cdr f3df))))
pr14 (cdr (assoc 13 (cdr (cdr f3df)))))
```

если точки pr1 и pr4 не совпадают:

```
(if (not (equal pr1 pr4))
```

введем новую систему координат с осью Z, проходящей через эти точки:

```
(command "ПСК" "zo" pr1 pr4)
```

и повернем весь оставшийся набор в выбранной системе координат на угол, вычисляемый через ссылку направления вектора, проходящего через начало координат и точку pr11 с конечным углом, соответствующим вектору с концом в точке pr2:

```
"ПОВЕРНИ" a "" '(0 0 0) "C"
(trans pr11 0 1) '(0 0 0)
(trans pr2 0 1)
```

Затем перейдем в мировую систему координат:

```
"ПСК" "М")
```

и удалим набор aa:

```
(command "СОТРИ" aa ""))
```

На этом шаг цикла заканчивается:

```
)
```

После выполнения всех поворотов посмотрим на полную картинку:

```
(command "покажи" "в")
```

Конец описания функции rslw:

```
)
```

Наметим пути усовершенствования программы. Прежде всего попробуйте расширить класс разворачиваемых поверхностей, учитывая, что каждая грань может соприкасаться не с одной следующей, а с двумя или тремя. Кроме того, если грань не плоская, ее следует разделить на две плоских.

2.3. Вставка блоков с относительными координатами

Блоки с атрибутами представляют значительное удобство для ввода параметров, которые изменяются при вставке в различные места чертежа. Однако возникают задачи, которые традиционными способами вставки блоков решить очень трудно. Например, требуется вставить ряд блоков через одну точку привязки, которая определяется в момент вставки первого блока. Традиционый путь - изменить систему координат при вставке первого блока, а затем задавать точки вставок последующих в новой системе координат.

Поступим несколько иначе. Для этого воспользуемся программой Ref, которая имеется в стандартном приложении к Автокаду. Несколько изменим ее так, чтобы предыдущая точка, записываемая в системную переменную LASTPOINT, не изменялась до окончания последнего ввода набора блоков, текста или других элементов чертежа:

```
(defun ref1 ()
  (if (= rpt nil)
  (setq rpt (setvar "LASTPOINT"
                    (getpoint "Точка ссылки: ")))
  )
  (getpoint "\nВведите относительные координаты (с @): ")
)
```

Предполагается, что в течение сеанса вставки группы блоков, масштабы по X, Y и Z не изменяются. Не изменяется и угол вставки блоков. Конечно, если это не устраивает, можно программу изменить и все необходимые действия запрограммировать.

Итак, сначала запомним соответствующие системные переменные и установим их так, чтобы они не мешали при работе новой команды:

```
(defun C:tinsert()
 (setq cmd (getvar "cmdecho")
      bpm (getvar "blipmode"))
```

```
(setvar "cmdecho" 0)
(setvar "blipmode" 0)
```

Затем можно поступить по-разному. Мы просто сформируем командную строку вызова команды ВСТАВЬ и включим ее в цикл, выход из которого обеспечивается при вводе nil на запрос очередной точки вставки в относительных координатах:

```
(setq namb (getstring "\nИмя блока: ")
    scx (getreal "\nX Масштаб <1>: "))
 (if (eq scx nil) (setq scx 1.0))
(setq scy (getreal "\nY Масштаб <1>: "))
 (if (eq scy nil) (setq scy 1.0))
(setq sca (getangle "\nУгол поворота <0>: "))
 (if (eq sca nil) (setq sca 0.0))
(setq rpt nil)
(while (not (eq (setq qpt (ref1)) nil))
   (command "Вставь" namb qpt scx scy sca)
   (setvar "lastpoint" rpt)
 )
```

Осталось вернуть исходные значения системным переменным и закончить программу:

```
        (setvar "cmdecho" cmd
        (setvar "blipmode" bpm)
        (princ)
)
```

Блок будет вставляться в относительных координатах до тех пор, пока вы не нажмете "ENTER" на запрос новой относительной точки ввода.

2.4. Программная вставка блоков с атрибутами

Попробуем решить более сложную задачу. Пусть теперь у нас блоки с атрибутами. Если атрибуты постоянны или вставляются через диалоговое окно, проблем не возникает. Просто перед вставкой блоков нужно установить системную переменную ATTDIA в единицу.

При вставке блоков с атрибутами, вводимыми через строку подсказки, потребуется в строку команды ВСТАВЬ добавлять слово "pause" столько раз, сколько у блока контролируемых атрибутов, да еще столько же "pause" на проверку правильности их ввода, т. е. удвоенное число количества контролируемых атрибутов, видимых или невидимых. Для определения числа атрибутов вставим "пробный" блок и проанализируем его атрибуты. Для подавления запросов о вставке атрибутов установим системную переменную ATTREQ в нуль. Затем подсчитаем число атрибутов, которые установились со значениями по умолчанию:

```
(setvar "attreq" 0)
(command "insert" namb (list 0 0) 0.01 0.01 sca)
```

Затем восстановим значение переменной ATTREQ и установим ATTDIA в нуль, а имя толко что вставленного блока запишем в переменную att:

```
(setvar "attreq" 1)
(setvar "attdia" 0)
(setq att (entlast))
(setq natt att)
```

Установим переменную подсчета требуемого числа пауз в нуль:

```
(setq npause 0)
```

Заготовим начало командной строки с именем блока, точкой вставки, которая будет вычисляться аналогично предыдущей задаче, масштабами вставки и углом поворота:

```
(setq comm '(command "insert" namb qpt scx scy sca))
```

Поскольку команда cons добавляет к строке спереди, обратим заготовку команды:

```
(setq comm (reverse comm))
```

Далее проанализируем все атрибуты данного блока:

```
(while (eq (cdr (assoc 0 (entget
        (setq att (entnext att)))))"ATTRIB")
```

Если блок видимый (его признак установлен в 4), тогда переменной с именем, записанным в имени атрибута, присваиваем значение атрибута, который является выражениям Автолиспа, предварительно освободившись от кавычек с помощью функции read, одновременно сосчитав требуемое для вставки данного сорта атрибутов число пауз:

```
(progn
(setq npause (1+ npause))
(if (eq (cdr (assoc 70 (entget att))) 4)
(set (read (cdr (assoc 2 (entget att))))
        (read (cdr (assoc 1 (entget att))))))
```

Если блок невидимый, в нем содержатся параметры, определяющие получаемую функцию, и их значение требуется вводить при каждой вставке блока. Получаемые вычисленные значения дописываем в список, выполнение которого обеспечит вставку требуемого блока:

```
(if (eq (cdr (assoc 70 (entget att))) 5)
(progn
(setq npause (1+ npause))
(princ "Attribute ")
(princ (cdr (assoc 1 (entget att))))
(princ)
(set (read (cdr (assoc 2 (entget att))))
```

```
    (getreal "\nInput attribute: "))))
(setq ata (eval (read (cdr (assoc 2 (entget att))))))
(setq comm (cons ata comm))))
```

Затем в список добавляем подсчитанное количество пауз:

```
(repeat npause
(setq comm (cons 'pause comm)))
```

Обращаем список в правильное положение:

```
(setq comm (reverse comm))
```

Включаем системную переменную CMDECHO в единицу для проведения диалога:

```
(setvar "cmdecho" 1)
```

И проводим вставку блока:

```
(eval comm)
```

Восстанавливаем системные переменные и заканчиваем задачу, удаляя вставленный предварительно блок:

```
        (setvar "lastpoint" rpt))
        (setvar "cmdecho" 0)
        (command "erase" natt "" "redraw")
        (setvar "cmdecho" cmd)
        (setvar "texteval" txx)
        (princ)
)
```

2.5. Блоки с памятью о последней вставке (создание основной надписи чертежа)

Формирование значений атрибутов по умолчанию происходит в момент их создания. Каждый раз при вставке нового блока с атрибутами вам предлагается набор атрибутов по умолчанию самого первого поколения. В некоторых ситуациях хотелось бы, чтобы в качестве значений по умолчанию предлагались значения, которые вводились в последнем сеансе вставки блока. Покажем, как это можно организовать при формировании основной надписи (штампа) чертежа. Заодно покажем, как можно использовать строку из меню ACAD.MNU "Формат" для задания поля чертежа, установки переменных для простановки размеров и создания основной надписи чертежа. Мы несколько модифицируем файл SETUP.LSP, добавив туда строки формирования основной надписи:

```
(apply '(lambda ()
(princ "\nРаботаю...")
(setq a (getvar "cmdecho"))
```

```
(setvar "cmdecho" 0)
(setq b (getvar "blipmode"))
(setvar "blipmode" 0)
(setvar "lunits" u)
(command
 "ЛИМИТЫ" "0,0" (list (* x c) (* y c))
 "ЛМАСШТАБ" (* 25 c)
 "РАЗМЕР" "РЗММАСШТ" c "ВЫХОД"
 "Отрезок" "0,0" (polar (getvar "lastpoint") 0 (* x c))
        (polar (getvar "lastpoint") (/ pi 2.0) (* y c))
        (polar (getvar "lastpoint")  pi (* x c))
         "Замкни"
 "Регенавто" "откл"  )
 (setq xl (* x c) yl (* y c))
 (cond ((and (equal u 1)(equal m 1.0))
        (setq x (rtos x 1 0) y (rtos y 1 0))
        (setq b "мм"))
      ((and (equal u 2)(equal m 1.0))
        (setq x (rtos x 2 0) y (rtos y 2 0))
        (setq b "мм"))
      ((and (equal u 1)(equal m 1000.0))
        (setq x (rtos x 1 3) y (rtos y 1 3))
        (setq b "м"))
      ((and (equal u 2)(equal m 1000.0))
        (setq x (rtos x 2 3) y (rtos y 2 3))
        (setq b "м"))
      ((equal u 3)
        (setq x (rtos x 3 2) y (rtos y 3 2))
        (setq b ""))
      ((equal u 4)
        (setq x (rtos x 4 2) y (rtos y 4 2))
        (setq b ""))  )
(setq a (strcat "\nВаш экран сейчас отображает область " x b "a " y b ". "))
 (setvar "coords" 2)
 (setq b c)
```

Эти строки почти дословно повторяют соответствующий текст стандартного файла SETUP.LSP 10-й версии Автокада.

Далее создадим стиль заполнения основной надписи штампа:

```
(command "стиль" "СТАНДАРТ" "italicc" 0 0.6 0 "н" "н" "н")
```

Теперь откроем текстовый файл, в который будет писаться содержимое вновь введенных атрибутов. Если такого файла нет, вставим блок с первоначальными значениями по умолчанию через диалоговое окно:

```
(setq tb (open "tbmag.txt" "r"))
(if (not tb)
(progn
(setvar "attdia" 1)
(command "Вставь" "preon" (list  xl 0) b "" 0))
```

Если же файл существует, установим системную переменную ATTDIA в нуль и в качестве вставляемых значений атрибутов используем содержимое файла:

```
(progn
(setvar "attdia" 0)
(while (not (setq tbmag (read (read-line tb)))))
(close tb)
(command "Вставь" "preon" (list xl 0) b "" 0 (nth 1 tbmag)
(nth 2 tbmag)(nth 3 tbmag)(nth 4 tbmag)(nth 5 tbmag)
(nth 6 tbmag)(nth 7 tbmag)(nth 8 tbmag)(nth 9 tbmag)
(nth 10 tbmag)(nth 11 tbmag)(nth 12 tbmag)(nth 13 tbmag)
(nth 1 tbmag)(nth 2 tbmag)(nth 3 tbmag)(nth 4 tbmag)
(nth 5 tbmag)(nth 6 tbmag)(nth 7 tbmag)(nth 8 tbmag)
(nth 9 tbmag)(nth 10 tbmag)
(nth 11 tbmag)(nth 12 tbmag)(nth 13 tbmag))
```

Для возможных изменений в перечне атрибутов вызовем команду "ДИАЛАТР":

```
(command "Диалатр" (entlast))))
```

Выделим из созданного блока значения атрибутов и запишем их заново в файл:

```
(setq blm (entlast))
(setq tb (open "tbmag.txt" "w"))
(setq tbmag '("Список данных штампа"))
(while (/= (cdr (assoc 0 (entget (entnext blm)))) "SEQEND")
(setq tbmag (cons (cdr (assoc 1 (entget
(entnext blm)))) tbmag))
(setq blm (entnext blm)))
(setq tbmag (reverse tbmag))
(prin1 tbmag tb)
(close tb)
```

Затем вставляется еще один блок в правом верхнем углу рамки чертежа и дорисовываются требуемые линии:

```
(command "Вставь" "2on" (list 0 yl) b "" 0 (nth 10 tbmag)(nth 10 tbmag))
(setq aa (list (- xl (* 5 b)) (* 60 b)))
(setq bb (list (- xl (* 5 b)) (- yl (* 5 b))))
(setq cc (list (* 20 b) (- yl (* 5 b))))
(command "плиния" (list (* 20 b) (* 5 b))
              "ш" (* 0.5 b) ""
              (list (- xl (* 190 b)) (* 5.0 b))  "")
(command "плиния" aa bb cc "")
(command "плиния" (list (* 20.0 b) (* 291.7978 b))
              (list (* 20.0 b) (- yl (* 5.0 b))) "")
(command "плиния" (list (* 20 b) (* 5 b)) (list (* 20 b)
(- yl (* 5.0 b))) "")
(princ a)
```

```
 (setq x nil y nil a nil aa nil bb nil cc nil
 b nil d nil c nil m nil u nil tbmag nil blm nil tb nil)
 (command "Регенавто" "вкл" "Покажи" "Все")
(princ))'())
```

2.6. Построение линий пересечения двух поверхностей

Решение этой задачи компенсирует некорректную работу команды Автокада СКРОЙ (HIDE), которая не обрабатывает пересекающиеся трехмерные грани с проведением линии их пересечения. Для получения более реалистической картины изображения перед выполнением команды СКРОЙ можно построить линии пересечения, которые проводятся в виде отрезков. В некоторых случаях эта задача полезна для выполнения сложных построений составных поверхностей. В 12-й версии Автокада команды РОК все это выполняют более изящно:

```
(defun pli (pq / phi dphi1 dphi2 dphi3 pin)
```

Определение принадлежности точки внутренности 3м грани, заданной точками p41, p42, p43 и p44:

```
(setq p41 (trans pr1 0 1)
p42 (trans pr2 0 1)
p43 (trans pr3 0 1)
p44 (trans pr4 0 1))
(setq phi (- (angle pq p41) (angle pq p44)))
(if (>= (abs phi) pi)
(setq phi (- phi (/ (* 2 pi (abs phi)) phi))))
(setq dphi1 (- (angle pq p42) (angle pq p41)))
(if (>= (abs dphi1) pi)
(setq dphi1 (- dphi1 (/ (* 2 pi (abs dphi1)) dphi1))))
(setq dphi2 (- (angle pq p43) (angle pq p42)))
(if (>= (abs dphi2) pi)
(setq dphi2 (- dphi2 (/ (* 2 pi (abs dphi2)) dphi2))))
(setq dphi3 (- (angle pq p44) (angle pq p43)))
(if (>= (abs dphi3) pi)
(setq dphi3 (- dphi3 (/ (* 2 pi (abs dphi3)) dphi3))))
(setq phi (+ phi dphi1 dphi2 dphi3))
(if (< (abs phi) 0.0001)
Если точка вне контура, pin=T
(setq pin T) (setq pin nil))
)

;Проведение линии пересечения
(defun crr (ps pe / nam)
(if (and (not (equal ps T))
(not (equal pe T)))
```

```
(progn
(setq f1 (pli ps))
(setq f2 (pli pe))
(command "ОТРЕЗОК" ps pe "")
(setq nam (entlast))
;Если концы линии выходят за контур, они обрезаются
(if (= f1 T)  (progn
(command "ОБРЕЖЬ" pll "" ps "")
(setq nam (entlast))))
(if (= f2 T)  (progn
(command "ОБРЕЖЬ" pll "" pe "")
(setq nam (entlast))))
;Если точки вне контура грани, линия удаляется
(setq pqs (cdr (assoc 10 (cdr (cdr (entget nam)))))
pqe (cdr (assoc 11 (cdr (cdr (entget nam)))))
pqs (trans pqs 0 1)
pqe (trans pqe 0 1))
(setq f1 (pli pqs))
(setq f2 (pli pqe))
(setq f3 (and f1 f2))
(if (= f3 T) (command "СОТРИ" nam ""))))
 )
(defun pc (ps pe / pcx pcy pcz pcr)
;Нахождение точки пересечения ребра и его проекции
;Если обе точки лежат по одну сторону грани, точка не определяется
;и ей присваивается значение T
(if (>= (* (caddr ps) (caddr pe)) 0.0)
(setq pcr T)
(progn
(setq pcx (+ (car ps) (/ (* (caddr ps) (- (car ps) (car pe)))
(- (caddr pe) (caddr ps))))
pcy (+ (cadr ps) (/ (* (caddr ps) (- (cadr ps) (cadr pe)))
(- (caddr pe) (caddr ps))))
pcz 0.0
pcr (list pcx pcy pcz))
 )))
(defun c:cross (/ fm sm a a1 la la1 st st1 pr1 pr2 pr3 pr4 nf3df f3df nf3df1
f3df1 pr11 pr12 pr13 pr14 p1 p2 p3 p4 p5 p6 p7 p8 pll)
(setq sblip (getvar "blipmode")
scmde (getvar "cmdecho"))
(setvar "blipmode" 0)
(setvar "cmdecho" 0)
(setq fm (car (entsel "\nУкажите первую сеть: "))
sm (car (entsel "\nУкажите вторую сеть: ")))
(command "слой" "c" "fm" ""
"слой" "c" "sm" ""
"свойства" fm "" "сл" "fm" ""
"свойства" sm "" "сл" "sm" ""
```

```
"РАСЧЛЕНИ" fm "РАСЧЛЕНИ" sm)
(setq a (ssget "x" '((0 . "3DFACE") (8 . "fm")))
la (sslength a)
a1 (ssget "x" '((0 . "3DFACE") (8 . "sm")))
la1 (sslength a1)
st 0)
(setvar "flatland" 0)
(repeat la
(command "ПСК" "М")
(setq st1 0
st (1+ st)
nf3df (ssname a (1- st))
f3df (entget nf3df)
aa f3df
pr1 (cdr (assoc 10 (cdr (cdr f3df))))
pr2 (cdr (assoc 11 (cdr (cdr f3df))))
pr3 (cdr (assoc 12 (cdr (cdr f3df))))
pr4 (cdr (assoc 13 (cdr (cdr f3df)))))
(if (and (not (equal pr1 pr4))
(not (equal pr2 pr4))
(not (equal pr1 pr2)))
(progn
(command "ПСК" "О" nf3df)
(repeat la1
(setq st1 (1+ st1)
nf3df1 (ssname a1 (1- st1))
f3df1 (entget nf3df1)
pr11 (cdr (assoc 10 (cdr (cdr f3df1))))
pr12 (cdr (assoc 11 (cdr (cdr f3df1))))
pr13 (cdr (assoc 12 (cdr (cdr f3df1))))
pr14 (cdr (assoc 13 (cdr (cdr f3df1)))))
(if (and (not (equal pr1 pr4))
(not (equal pr2 pr4))
(not (equal pr1 pr2)))
(progn
(command "слой" "с" "3" "ц" "4" "" "")
(setq p1 (trans pr11 0 1)
p2 (trans pr12 0 1)
p3 (trans pr13 0 1)
p4 (trans pr14 0 1)
p5 (pc p1 p2)
p6 (pc p2 p3)
p7 (pc p3 p4)
p8 (pc p4 p1))
(command "плиния" (trans pr1 0 1)
"ш" 0 0 (trans pr2 0 1)
(trans pr3 0 1)
(trans pr4 0 1) "замкни")
(setq pll (entlast))
```

```
(crr p5 p6)
(crr p5 p7)
(crr p5 p8)
(crr p6 p7)
(crr p6 p8)
(crr p7 p8)
(command "СОТРИ" pll "" "ОСВЕЖИ")
          ))))
          )
          )
(command "СЛОЙ" "с" "1" "" "ОСВЕЖИ")
(setvar "blipmode" sblip)
(setvar "cmdecho" scmde)
)
```

Написав на Автолиспе программу, содержащую много раз выполняемый цикл, вы тут же убедитесь в том, что Автолисп - медленный язык. Это связано с тем, что вычислитель Автолиспа - это интерпретатор, и в этом смысле подобен Бейсику. Иначе говоря, программа на Автолиспе - это, собственно, не программа, а данные для вычислителя Автолиспа. Выполняя, например, цикл, вычислитель Автолиспа многократно интерпретирует одни и те же строки программы, не "понимая" того, что эта работа уже была один раз проделана. Чтобы убедиться в этом, загрузите и выполните программу на Автолиспе fplot, входящую в комплект стандартной поставки Автокада. Если вы работаете только с функциями Автокада и вводом/выводом данных, то скорость вычислений не особенно критична, но, как только возникнет необходимость в большом объеме вычислений, Автолисп перестанет вас удовлетворять.

Значит ли это, что Автолисп исчерпал свои возможности? Оказывается, нет. Этот недостаток Автолиспа 10-ой версии устранен с появлением компилятора Автолиспа. Какие же преимущества имеет откомпилированная программа на Автолиспе по сравнению с обычной?

- В 15-20 раз повышается быстродействие, как за счет собственно отказа от интерпретации программ, так и за счет увеличения быстродействия постраничной виртуальной памяти.
- Загрузка программ в память происходит втрое быстрее, поскольку теперь при загрузке программы не требуется проверять ее на предмет наличия синтаксических ошибок.
- В среднем вдвое уменьшаются требования к объему памяти как для кода программ, так и для данных - компилированные функции могут содержать больше данных.
- Компилятор обеспечивает работу как со стандартным, так и с расширенным Автолиспом. Компилированные и интерпретированные функции могут комбинироваться как угодно и вызывать друг друга. Большинство имеющихся в настоящее время приложений на Автолиспе можно откомпилировать без каких-либо изменений, обес-

печив тем самым резкое увеличение скорости выполнения как самых простых, так и наиболее сложных специализированных программ.

- Используя новые возможности отладки программ при работе с расширенным Автолиспом, вы можете прервать выполнение сложной программы в любом месте, просмотреть или изменить значения переменных, выполнить некоторые отладочные операции и продолжить работу.
- Как отметил Фил Ротуэлл, один из разработчиков системы Автокад, применение компилятора позволит вам не беспокоиться об авторских правах - защита ваших программ обеспечена.

Использование в 11-й и 12-й версиях Автокада разработок на Си снимает необходимость использования компилятора Автолиспа, поскольку для коммерческих и быстродействующих приложений можно воспользоваться именно этим способом разработки приложений. Поэтому для Автолиспа в 11-й и 12-й версиях компилятор не разрабатывался.

Здесь мы только коснулись поверхности океана возможностей, предоставляемых Автокадом, описали логику его построения и использование Автолиспа в работе с базой данных Автокада. Для более глубокого изучения Автолиспа обращайтесь к учебникам программирования на Автолиспе, выпущенным издательством Addison Wesley, - например, к книге Winston and Horn "LISP" (second edition) или прекрасному учебнику T. Hasemer "Looking at LISP". В качестве полезного справочника по Автолиспу можно рекомендовать также книгу "Автолисп. Руководство по программированию", изданную на русском языке.

Вот некоторые полезные приемы, использование которых является признаком "хорошего стиля" работы на Автокаде:

- Используйте Автолисп для создания своих макроопределений меню, более сложных и "умных", чем стандартные.
- По возможности вычисляйте значения параметров, старайтесь реже запрашивать пользователя ввести ту или иную величину - это уменьшит вероятность ошибок при вводе.
- Старайтесь, чтобы создаваемые макроопределения содержали как можно больше подсказок - это также помогает уменьшить вероятность ошибок.
- Используйте функции GET для ввода данных.
- Старайтесь без необходимости не объявлять переменные глобальными - это засоряет память, создает дополнительную путаницу, уменьшает скорость выполнения функций.
- Для использования функций Автокада применяйте стандартную функцию COMMAND.

- Помните, что, используя в функции defun объявление C:, вы можете расширить список команд Автокада.
- Используйте функции AND/OR для переключения вывода на экран разных сообщений по условию.
- Старайтесь защититься от неправильного ввода функцией GETKWORD или, если это невозможно, организуйте цикл WHILE.
- Делайте вашу программу нечувствительной к разнице между строчными и прописными буквами, используя функцию OR или STRCASE.

Диалоговые окна в Автокаде

При создании достаточно сложных программ ввод исходной информации через командную строку не всегда бывает наглядным и удобным. Ведение диалога с программой можно усовершенствовать, используя средства самого Автокада: здесь в нашем распоряжении графические меню, средства редактирования атрибутов и диалоговые окна. Для 10-ой и 11-ой версий Автокада применение диалоговых окон было ограничено либо созданием вспомогательных программ, описываемых как внешние функции с последующей передачей параметров через текстовые файлы, читаемые командами Автолиспа, либо программами на Автолиспе, управляющими текстовым режимом экрана через драйвер ansi.sys передачей ESC-последовательностей, управляющих фоном текста и заполнением строк требуемой информацией. Примером такой программы может служить программа предварительного ввода данных в диалоге на текстовом экране:

```
(TEXTSCR)
(DEFUN MENU-OPERATION (HEADER ITEM-LIST PRMPT COLOR / HGT WDT I L-COL)
(MENU_INIT color1)
(PAINT_BKGRND TOP_MARG L_COL HGT WDT color1)
(PAINT_FRAME TOP_MARG L_COL HGT WDT)
(PRINT_HEADER TOP_MARG L_COL WDT)
(PRINT_ITEMS ITEM-LIST TOP_MARG L_COL color1)
(PRINT_PRMPT PRMPT TOP_MARG L_COL HGT)
(USR_VAL))
(DEFUN MENU_INIT (COLOR)
(TEXTSCR)
(CLS)
(NORMAL)
(PRINC (STRCAT "\e[" (ITOA COLOR) "m"))
(IF (/= (REM (STRLEN HEADER) 2) 0)
(SETQ HEADER (STRCAT HEADER " ")))
(SETQ HGT (+ 5 (LENGTH ITEM-LIST))
WDT (+ 10 (MAX (LONGEST ITEM-LIST) (STRLEN HEADER))))
(IF (/= (REM HGT 2) 0) (SETQ HGT (1+ HGT)))
(IF (/= (REM WDT 2) 0) (SETQ WDT (1+ WDT)))
(SETQ L_COL (- 40 (/ WDT 2))
i 0
TOP_MARG (- 12 (/ HGT 2))))
(DEFUN PAINT_BKGRND (RW CL HT WD COLOR)
(IF (> COLOR 40)
(PROGN (GOTO (1+ RW) (1+ CL))
```

```
(REPEAT (- HT 1)
(REPEAT (- WD 2) (PRINC " " ))
(NEXTROW (- WD 2))))))
(DEFUN PAINT_FRAME (RW CL HT WD)
(GOTO RW CL)
(PRINC (CHR 201))
(REPEAT (- WD 2)
(PRINC (CHR 205)))
(PRINC (CHR 187))
(REPEAT 3
(NEXTROW WD)
(PRINC (CHR 186))
(MOVE (- WD 2) "C")
(PRINC (CHR 186)))
(NEXTROW WD)
(PRINC (CHR 204))
(REPEAT (- WDT 2) (PRINC (CHR 205)))
(PRINC (CHR 185))
(REPEAT (- HT 5)
(NEXTROW WD)
(PRINC (CHR 186))
(MOVE (- WD 2) "C")
(PRINC (CHR 186)))
(NEXTROW WD)
(PRINC (CHR 200))
(REPEAT (- WDT 2) (PRINC (CHR 205)))
(PRINC (CHR 188)))
(DEFUN PRINT_HEADER (RW CL WD)
(GOTO (+ RW 3)
(+ CL (- (/ WD 2) (/ (STRLEN HEADER) 2))))
(BOLD)
(PRINC HEADER)
(NORMAL))
(DEFUN PRINT_HEADER (RW CL WD)
(GOTO (+ RW 2)
(+ CL (- (/ WD 2) (/ (STRLEN HEADER) 2))))
(BOLD)
(PRINC HEADER))
(DEFUN PRINT_ITEMS (ITM_LST RW CL COLOR)
(PRINC (STRCAT "\e[0m\e[" (ITOA COLOR) "m"))
(SETQ I 0)
(FOREACH ITEM ITM_LST
(SETQ I (1+ I))
(GOTO (+ RW 4)
(+ CL 2))
(MOVE I "B")
(PRINC (STRCAT " "
(IF (< I 10) " " "")
(RTOS (FLOAT I) 2 0) "] " ITEM))))
```

```
(DEFUN PRINT_PRMPT (PRMPT RW CL HT)
(NORMAL)
(GOTO (+ RW HT 3) 0)
(PRINC PRMPT)
(GC))
(DEFUN USR_VAL ()
(NORMAL)
(SETQ CHOICE (GETINT))
(WHILE (OR (< CHOICE 1) (> CHOICE (LENGTH ITEM-LIST)))
(SETQ CHOICE (GETINT "Выбор вне диапазона, повторите: ")))
(CLS)
(LIST CHOICE (nth (1- CHOICE) ITEM-LIST)))
;;Длина самой длинной строки определяет ширину таблички
(DEFUN LONGEST (LST)
(APPLY 'MAX (MAPCAR '(LAMBDA (ITM) (STRLEN ITM)) LST)))
(DEFUN BOLD ()
(PRINC "\e[1m"))
(DEFUN NORMAL ()
(PRINC "\e[0m"))
```

Группа этих подпрограмм формирует рамку с бордюрами на текстовом экране для заполнения ее строками с альтернативным выбором соответствующего номера, который затем анализируется и используется на последующих этапах решения общей задачи.

Функция A1:

```
(defun A1 ()
(setq color1 41)
(setq aa1
(menu-operation "Выбор типа изделия"
'("Форма по ГОСТ 2209-82 типа 06"
"Форма по ГОСТ 2209-82 типа 16"
"Форма по ГОСТ 2209-82 типа 07"
"Форма по ГОСТ 2209-82 типа 38"
"Форма по ГОСТ 19042-82 и ТУ 48-19-307-86 типа V, C, D"
"Форма по ГОСТ 19042-82 и ТУ 48-19-307-86 типа W")
"Выберите соответствующий тип для проекта: "
(ran_color) )))
```

Несмотря на ограниченность такого подхода к формированию диалоговых окон, они оказались весьма полезными во многих прикладных задачах.

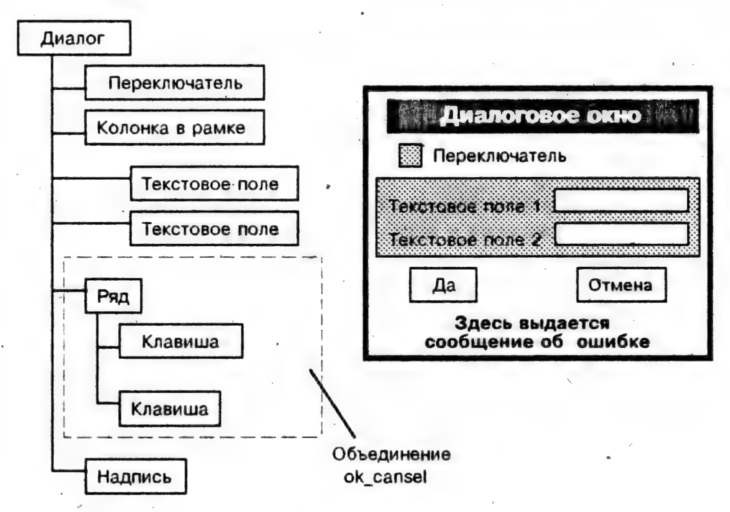
В 12-й версии Автокада число диалоговых окон в самом графическом редакторе резко расширилось и появилась целая система программного формирования диалоговых окон - язык управления диалогом DCL вместе с комплексом команд Автолиспа, дающих возможность вызова и управления диалогом из Лисп-программ. Все это значительно расширяет возможности адаптации Автокада под конкретные прикладные задачи.

3.1. Язык управления диалогом - DCL

Диалоговые окна в 12-й версии Автокада определяются текстовыми файлами, написанными на DCL. Эти файлы имеют расширение .dcl. В них содержится описание способа вывода окна и его состав: клавиши, списки, скользящие шкалы, кнопки выбора и т. п. Правила конструирования диалоговых окон задают ограничения на размер и место расположения окна. Расположение элементов окна очень похоже на расположение абзацев в сформатированном тексте, поэтому нет необходимости задавать точные координаты фрагментов окон.

Диалоговое окно из меню Автокада вызывается через функцию Автолиспа или внешнюю функцию СРП (Средства Разработки Приложений), которые управляют диалоговым окном.

В каждом диалоговом окне содержится одно или несколько полей, определяющих функции окна. К основным типам полей относятся базовые поля: клавиши, кнопки, текстовые поля, скользящие шкалы, поля списков, поля изображений. Поля могут обрамляться рамками. Можно комбинировать поля, создавая ряды и колонки. Каждое диалоговое окно рассматривается как древовидная структура, вершина которой представляется на языке DCL как dialog. Управление появлением и поведением поля на этом языке осуществляется атрибутами поля. Можно определить новые поля (прототипы) и группы полей, которые не связаны с обыкновенными диалоговыми окнами. На прототипы можно ссылаться и изменять при необходимости их атрибуты и предварительно определенные поля. Объединения используются только для внешних ссылок. Их атрибуты изменять нельзя.



На этом рисунке показан пример структуры диалогового окна. Уровни дерева - это встроенные поля, а вершина дерева - примитив диалога. Описание диалогового окна дается на языке DCL, отражающем его древовидную структуру.

Предварительно определенные активные поля

Предварительно определенные поля непосредственно поддерживаются средствами программируемых диалоговых окон Автокада. Их определения содержатся в файле *base.dcl* в виде комментариев. При выборе активного поля диалоговое окно извещает об этом приложение (программу на Лиспе или Си), управляющее диалоговым окном. Подобная операция называется ***действием*** или ***вызовом с возвратом.*** Любое предварительно определенное активное поле имеет соответствующий видимый внешний эффект (например, раскрытие списка или закрытие диалога при указании клавиши "ДА") или внутренний эффект. В этом случае вырабатывается код причины, смысл которого зависит от типа поля, его инициировавшего.

► Тип поля

Клавиша - **button**: Поле диалогового окна, напоминающее обычную клавишу клавиатуры. Клавиши предназначены для выполнения действий, имеющих внешний эффект для пользователя: закрытие диалогового окна, появление еще одного окна и т. д. Все диалоговые окна содержат клавишу "Да" или ее эквивалент, позволяющую выполнить действие окна, а многие окна - клавишу "Отмена", позволяющую пользователю покинуть диалоговое окно без внесения каких-либо изменений.

Атрибуты

label: Метка - строка, заключенная в кавычки (значения по умолчанию нет. Определяет надпись, которая появляется на клавише.

is_default: Возможные значения: true или false (по умолчанию - false). Если значение true, клавиша является клавишей по умолчанию и указывается, когда пользователь нажимает клавишу "RETURN". Если пользователь работает в текстовом поле, поле списка или клавише изображения, имеющих атрибут allow_accept, установленный в true, клавиша по умолчанию также выбирается путем указания клавиши принятия или (для полей списка и клавиш изображения) выполнения двойного указания с помощью кнопки выбора устройства указания. Клавиша по умолчанию не выбирается при нажатии клавиши

выполнения, если подсвечена другая клавиша: в этом случае будет выбрана подсвеченная клавиша.

Клавиша по умолчанию выводится на экран отличным от других клавиш способом, например с помощью дополнительной рамки.

Только одна клавиша в диалоговом окне может иметь значение атрибута is_default, равное true.

is_cancel: Возможные значения: true или false (по умолчанию - false). Если значение true, клавиша выбирается при нажатии клавиши отмены (например, ESC или Ctrl+C).

Только одна клавиша в диалоговом окне может иметь значение атрибута is_cancel, равное true. Клавиша, имеющая установленный в true атрибут is_cancel, закрывает диалоговое окно после выполнения действия или вызова с возвратом.

► Тип поля

Текстовое поле - edit_box: Поле в котором пользователь может вводить или редактировать текстовую строку. Если вводимый текст превышает размеры текстового поля, его можно "прокрутить" в горизонтальном направлении.

Атрибуты

Label: Значение - строка, заключенная в кавычки (по умолчанию - пустая - " "). Текст выводится слева от поля. Если он определен, атрибут выравнивается влево по ширине текстового поля.

Edit_width: Возможные значения: целое или вещественное число. Ширина редактируемой части поля, ограниченной рамкой текстового поля в единицах ширины символ. Если атрибут не указан или установлен в нуль и ширина поля не ограничена, рамка расширяется настолько, насколько это возможно. Если значение атрибута не равно нулю, рамка выравнивается вправо внутри занятого полем пространства. Если необходимо вытянуть поле, можно при компоновке вставить пробелы между атрибутом и редактируемой частью поля средствами PDB.

Edit_limit: Значение - целое число (по умолчанию - 132); максимум - 256. Максимальное число символов, которое может ввести пользователь в текстовое поле. Когда этот предел достигается, драйвер монитора отбрасывает дополнительные символы (кроме Backspase или Del) и выдает сигнал.

Value: Значение - строка, заключенная в кавычки (по умолчанию - пустая - " "). Исходное значение текстовое, размещенное в рамке. Значение выравнено влево в редактируемой части поля.

Значение текстового поля всегда заканчивается пустым вводом (\0 или EOS). Если пользователь вводит больше символов, чем установлено атрибутом edit_limit, и при этом необходимо усечь строку, добавляется пустой символ.

Allow_accept: Возможные значения: true или false (по умолчанию - false). Если значение равно true и пользователь нажал клавишу выполнения (обычно клавиша "RETURN", клавиша по умолчанию (если есть) становится "нажатой". (Клавишей по умолчанию считается клавиша, атрибут is_default которой установлен в true.

▶ Тип поля

Клавиша изображения - image_button: Поле с нарисованным графическим изображением. Когда пользователь выбирает клавишу изображения, программа получает координаты точки, в которой произошел выбор. Это может быть полезно, если нарисован небольшой рисунок и выбор разных его областей имеет разное значение.

Атрибуты

Color: Цвет фона рисунка, определенный как номер цвета Автокада или одно из символьных имен (по умолчанию - 7):
dialog_line - текущий цвет диалогового окна;
dialog_forground - текущий цвет символов диалогового окна (для текста);
dialog_background - текущий цвет фона диалогового окна;
graphic_background - текущий цвет фона графического экрана Автокада (обычно аналог 0);
black - цвет Автокада (черный) на черном фоне изображается как белый;
red - цвет Автокада 1 (красный);
yellow - цвет Автокада 2 (желтый);
green - цвет Автокада 3 (зеленый);
cyan - цвет Автокада 4 (голубой);
blue - цвет Автокада 5 (синий);
magenta - цвет Автокада 6 (фиолетовый);
white - цвет Автокада 7 (белый);
graphic_forground на белом фоне изображается как черный.

Allow_accept: Возможные значения: true или false (по умолчанию - false). Если значение равно true и пользователь нажал клавишу выполнения (обычно клавиша "RETURN"), клавиша по умолчанию (если есть) становится "нажатой". (Клавишей по умолча-

нию считается клавиша, атрибут is_default которой установлен в true.

Acpect_ratio: Отношение ширины изображения к его высоте (ширина, деленная на высоту). Если значение равно нулю (0.0), то поле подгоняется под размеры изображения. Возможно значение с плавающей точкой (значения по умолчанию нет).

Полю изображения необходимо присвоить точные значения атрибутов width и height или один из этих атрибутов плюс значение acpect_ratio.

► Тип поля

Поле списка - list_box Поле, содержащее ряды текстовых строк. Наличие такого поля дает возможность пользователю выбрать необходимый пункт из списка. Обычно список имеет переменную длину, но поля списков могут использоваться и для списков фиксированной длины. При выборе строки списка она подсвечивается. Поле списка может содержать строк больше, чем вмещает поле; в этом случае справа от списка поля появляется скользащая шкала (она включена только тогда, когда список имеет больше пунктов, чем можно ввести за один раз). Перемещая визир скользящей шкалы или указывая на стрелки, можно просмотреть весь список. В зависимости от приложения пользователь может выбрать несколько строк списка.

Атрибуты

Label: Текст, выводимый над полем списка. Значение - строка, заключенная в кавычки (значения по умолчанию - нет).

Maltiple_select: Возможные значения true или false (по умолчанию - false). Если true, в поле списка можно выбрать (и подсветить несколько пунктов. Если false, в поле списка может быть выбран только один пункт, а выбор другого пункта отменит выбор предыдущего.

List: Определяет начальный выбор строк, размещенных в поле списка. Строки разделяются символом новой строки (\n). Символ табуляции (\t) может присутствовать в каждой строке. Значение - строка, заключенная в кавычки (значения по умолчанию нет).

Tabs: Значение - строка, заключенная в кавычки, содержащая целые числа или числа с плавающей точкой, разделенные пробелами (значения по умолчанию нет). Каждое число - это величина, которая определяет расположение табуляций в единицах ширины символа. Эти значения используются для вертикального выравнивания колонок текста в поле списка (см. диалоговое

окно управления слоями Автокада, acad_mylayer в файле acad.dcl).

Value: Значение - строка, заключенная в кавычки, может содержать нуль ("0") или целые числа, разделенные пробелами (значения по умолчанию нет). Каждое целое (начиная с нуля) представляет первоначально выбранный пункт списка. Если значение атрибута multiple_select равно false, атрибут value не может содержать более одного числа.

Если строка пуста (" "), первоначально не будет выбран ни один пункт. В этом случае этот атрибут можно вообще не определять.

Allow_accept: Возможные значения: true или false (по умолчанию - false). Если значение равно true и пользователь нажал клавишу выполнения (обычно клавиша "Return"), клавиша по умолчанию становится "нажатой". (Клавишей по умолчанию считается клавиша, атрибут is_default которой установлен в true.

► Тип поля

Раскрывающийся список - popup_box Раскрывающийся список эквивалентен полю списка. Когда диалоговое окно появляется первый раз, раскрывающийся список находится в закрытом состоянии и на экране он выглядит скорее как клавиша с направленной вниз стрелкой справа. При выборе надписи или стрелки список раскрывается и позволяет выбрать необходимый пункт в пределах окна. Раскрытый список, как правило, отображается полностью, в противном случае он будет иметь скользящую шкалу справа, которая функционально аналогична скользящей шкале поля списка. Когда раскрывающийся список закрыт, текущий выбор появляется в его видимой области. Раскрывающиеся списки не допускают выбора нескольких элементов одновременно.

Атрибуты

Label: Значение - строка, заключенная в кавычки (значения по умолчанию нет). Это текст, выводимый слева от раскрывающегося списка. Если значение атрибута label определено, оно выравнивается влево в пределах поля popup_list.

Edit_width: Возможные значения: целое или вещественное число. Ширина части списка в единицах ширины символа, ширина рамки, описывающей единственный пункт, когда раскрывающийся список закрыт. Список не включает необязательную метку слева и стрелку (или скользящую шкалу) справа. Если атрибут не указан или установлен в нуль и ширина поля не за-

дана, рамка расширяется настолько, насколько это возможно. Если значение атрибута не равно нулю, рамка выравнивается вправо внутри занятого полем пространства. Если необходимо вытянуть поле, можно при компоновке вставить пробелы между атрибутом и редактируемой частью поля средствами PDB.

Value: Значение - строка, заключенная в кавычки, содержащая целое число (по умолчанию 0). Целое число (начиная с нуля) представляет текущий выбранный пункт в списке (пункт, который выводится на экран при закрытом списке).

List: Определяет начальный выбор строк, размещенных в поле списка. Строки разделяются символом новой строки (\n). Символ табуляции (\t) может присутствовать в каждой строке. Значение - строка, заключенная в кавычки (значения по умолчанию нет).

Tabs: Значение - строка, заключенная в кавычки, содержащая целые числа или числа с плавающей точкой, разделенные пробелами (значения по умолчанию нет). Каждое число - это величина, которая определяет расположение табуляций в единицах ширины символа. Эти значения используются для вертикального выравнивания колонок текста в раскрывающемся списке.

► Тип поля

Кнопка выбора - radio_button Одна или группа кнопок, объединенных в колонку или ряд выбора. Кнопки выбора функционально эквивалентны кнопкам радиопанелей: можно выбрать одну кнопку, и, пока она будет в нажатом состоянии, любая другая кнопка в колонке (ряду) будет отключена. Справа от кнопки выбора может появиться необязательная метка (label). Кнопки выбора появляются только в колонках или рядах выбора. Если кнопка выбора размещена вне колонки или ряда выбора, сообщается об ошибке.

Атрибуты

Label: Это текст, выводимый слева от клавиши выбора. Значение - строка, заключенная в кавычки (значения по умолчанию нет).

Value: Заключенная в кавычки строка (значения по умолчанию нет). Если значение атрибута единица, кнопка выбора включена; если нулю - кнопка выбора отключена; все другие значения аналогичны нулю.

Если по каким-то причинам несколько кнопок имеют атрибут value, равный единице, будет включена только последняя из них. (Эта ситуация может иметь место только в пользовательских DCL-файлах. Как только диалоговое окно появилось

на экране, средства PDB управляют кнопками выбора и добиваются, чтобы могла быть включена только одна кнопка в группе одновременно.)

▶ Тип поля

Скользящая шкала - slider Подразумевает получение численного значения. Пользователь может перемещать визир скользящей шкалы влево или вправо (или вверх и вниз) для получения величины, назначение которой зависит от приложения. Эта величина возвращается как строка, содержащая целое число определенной точности со знаком. В приложении это значение можно масштабировать.

Атрибуты

Min_value и **Max_value** Значение - целые числа, определяющие диапазон возвращаемых скользящей шкалой значений. Минимальное значение по умолчанию, min_value - 0. Максимальное значение по умолчанию max_value - 10000. Диапазон должен быть определен знаковым 16-битовым целым, т. е. от -32768 до 32767.

Значение атрибута min_value может быть больше, чем значение max_value. На некоторых платформах это изменяет как последовательность появления на экране этих значений, так и последовательность их возврата скользящей шкалой во время перемещения визира.

Small_increment и **big_increment** Значение - целые числа, определяющие значения, используемые при управлении приращением значения скользящей шкалы. Значение по умолчанию для атрибута big_increment - одна десятая полного диапазона, значение по умолчанию для small_increment - сотая от полного диапазона. Значения должны быть в диапазоне, определяемом атрибутами min_value и max_value. Эти атрибуты не обязательны.

Layout: Скользящая шкала может быть ориентирована горизонтально или вертикально (по умолчанию - горизонтально). Для горизонтальных шкал значение увеличивается слева направо, для вертикальных - снизу вверх.

Value: Значение - строка, заключенная в кавычки и содержащая текущее (целое) значение скользящей шкалы (по умолчанию - min_value).

▶ Тип поля

Переключатель - tuggle

Переключатель оперирует двоичными величинами (“0” или “1”). Он изображается небольшим прямоугольником с необязательной меткой (label) справа. Метка X появляется или исчезает в прямоугольнике при указании переключателя. Переключатель позволяет пользователю видеть и изменять его состояние “включен_выключен”.

Атрибуты

Label: Текст, выводимый слева от кнопки переключателя. Значение - строка, заключенная в кавычки.

Value: Определяет первоначальное состояние переключателя. Значение - строка, заключенная в кавычки и содержащая целое число (по умолчанию - нуль). Если значение равно нулю, кнопка переключателя пуста (не помечена). Если значение равно единице, кнопка выводится с меткой X.

Предварительно определенные активные группы полей

Поля можно группировать в объединенные ряды или колонки. Сгруппированные ряды и колонки при формировании общего диалогового окна рассматриваются как единое поле и могут быть заключены з рамку и иметь необязательную метку (группа без рамки не может иметь метки). Однако при выборе группа не может иметь присвоенное ей действие (кроме ряда или колонки выбора) и представляет собой просто удобный способ раскладки полей в диалоговом окне. Ряд или колонку можно определить для внешнего использования как элемент диалогового окна. Группы, используемые подобным образом, называются объединениями, так как они могут содержать другие поля (детей). Нельзя изменять атрибуты объединения, если оно используется как ссылка в диалоговом окне. В файле base.dcl определено несколько стандартных объединений.

▶ Тип поля

Колонка - column Колонка или любой вид поля (кроме кнопки выбора), включая ряды и другие колонки. Все элементы колонки располагаются вертикально в порядке расположения в DCL-файле.

Атрибуты

Колонка без рамки не имеет дополнительных атрибутов, кроме стандартных атрибутов компоновки.

► Тип поля

Колонка в рамке - boxed-column Колонка, имеющая нарисованную вокруг нее рамку (границу). Диалоговое окно расположено, как и колонка, в рамке. Если у колонки в рамке имеется метка, она может появиться выше рамки или быть встроенной в нее. Если метка отсутствует или является пробелом (" "), или пусто (""), изображается только рамка.

Атрибуты

Label: Значение - строка, заключенная в кавычки (по умолчанию " "). Атрибут label выводится как поле в верхнем левом углу колонки, заключенной в рамку.

► Тип поля

Ряд - row Аналогичен колонке, но его поля расположены горизонтально в порядке расположения в DCL-файле.

Атрибуты

Ряд без рамки не имеет дополнительных атрибутов, кроме стандартных атрибутов компоновки.

► Тип поля

Ряд в рамке - boxed_row Ряд, имеющий нарисованную вокруг него рамку (границу). Если у рядав рамке имеется метка, она может появиться выше рамки или быть встроенной в нее. Если метка отсутствует или является пробелом (" ") или пусто (""), изображается только рамка.

Атрибуты

Label: Значение - строка, заключенная в кавычки (по умолчанию " "). Атрибут label выводится как поле в верхнем левом углу ряда, заключенного в рамку.

► Тип поля

Колонка выбора - radio_column Колонка, содержащая поля кнопок выбора. За один раз можно выбрать только одну из кнопок. Это фиксированный набор взаимоисключающих альтернатив. В отличие от обычных колонок колонкам выбора можно присвоить действие.

Атрибуты

Value: Значение - строка, заключенная в кавычки, содержащая значение ключа (key) выбранной текущей кнопки выбора.

▶ Тип поля

Колонка выбора в рамке - boxed_radio_column Колонка выбора, имеющая нарисованную вокруг нее рамку (границу). Метка трактуется аналогично метке колонки в рамке.

Атрибуты

Label: Значение - строка, заключенная в кавычки (по умолчанию " "). Атрибут label выводится как поле в верхнем левом углу колонки, заключенной в рамку.

Value: Значение - строка, заключенная в кавычки, содержащая значение ключа (key) выбранной текущей кнопки выбора.

▶ Тип поля

Ряд выбора - radio_row Аналогичен ряду выбора и содержит поля кнопок выбора, но при этом за один раз можно выбрать только одну кнопку. Ряду выбора можно присвоить действие.

Атрибуты

Value: Значение - строка, заключенная в кавычки, содержащая значение ключа (key) выбранной текущей кнопки выбора.

▶ Тип поля

Ряд выбора в рамке - boxed_radio_row Ряд выбора, заключенный в рамку (границу). Метка трактуется как и метка колонки в рамке.

Атрибуты

Label: Значение - строка, заключенная в кавычки (по умолчанию " "). Атрибут label выводится как поле в верхнем левом углу ряда, заключенного в рамку.

Value: Значение - строка, заключенная в кавычки, содержащая значение ключа (key) выбранной текущей кнопки выбора.

Декоративные и информационные поля

▶ Тип поля

Изображение - image Прямоугольник, внутри которого отображается векторное изображение.

Атрибуты

Color: Цвет фона рисунка, определенный как номер цвета Автокада или одно из символьных имен (по умолчанию - 7):
dialog_line - текущий цвет диалогового окна;

dialog_forground - текущий цвет символов диалогового окна (для текста);
dialog_background - текущий цвет фона диалогового окна;
graphic_background - текущий цвет фона графического экрана Автокада (обычно аналог 0);
black - цвет Автокада (черный) на черном фоне изображается как белый;
red - цвет Автокада 1 (красный);
yellow - цвет Автокада 2 (желтый);
green - цвет Автокада 3 (зеленый);
cyan - цвет Автокада 4 (голубой);
blue - цвет Автокада 5 (синий);
magenta - цвет Автокада 6 (фиолетовый);
white - цвет Автокада 7 (белый).
graphic_forground на белом фоне изображается как черный

Acpect_ratio: Отношение ширины изображения к его высоте (ширина, деленная на высоту). Если значение равно нулю (0.0), то поле подгоняется под размеры изображения. Возможно значение с плавающей точкой (значения по умолчанию нет).

Полю изображения необходимо присвоить точные значения атрибутов width и height или один из этих атрибутов плюс значение acpect_ratio.

► Тип поля

Надпись - text Текстовая строка, используемая для вывода заголовка поля и диалогового окна или с информационной целью.

Атрибуты

Label: Отображаемый на экране текст. Значение - строка, заключенная в кавычки (значения по умолчанию нет). При компоновке поля надписи его ширина больше значения атрибута width, определенного в DCL-файле, или ширины, определяемой атрибутом label, если он задан. Если ни один из них не задан, выдается сообщение об ошибке.

Value: Как и label, определяет строку, выводимую в поле надписи, но не влияет на компоновку полей. Если сообщение является неизменяемым, следует определить атрибут label и не определять width и value. В противном случае определяется атрибут value и присвоить атрибуту width присваивается достаточно большое значение. После вывода диалогового окна на экран нельзя изменить размер его полей; если при выводе функции set_tile надписи присвоено более длинное значение, чем ширина окна, надпись будет усечена.

Is_bold: Возможные значения: true или false (по умолчанию - false). Если true, надпись выводится полужирным шрифтом.

▶ Тип поля

Разделитель - spaser Пустое поле. Используется только в целях компоновки и влияет на размер и расположение смежных полей.

Атрибуты

Нет.

В этих таблицах показаны поля, используемые в диалоговых окнах вместе с их атрибутами. Атрибуты полей определяют их размещение и функциональность. Атрибут похож на переменную языка программирования. Он состоит из имени и значения. Значения атрибутов могут быть:

- целыми - это численная величина (целая или вещественная), которая представляет собой размер в единицах ширины или высоты символов;
- вещественными - это численная величина, в которой наличие незначащего нуля в целой части обязательно;
- строковыми - это строка текста, заключенная в кавычки (“ ”). Если строка содержит символ кавычки, перед ним следует поставить обратную косую черту: \”. Строка может содержать другие ESC-последовательности:

  \” - кавычку;
  \\ - обратную косую черту;
  \n - новую строку;
  \t - горизонтальную табуляцию;

- зарезервированными словами - это идентификатор, состоящий из текстовых символов и начинающийся с буквы (например, true или false - зарезервированные слова, требуемые во многих атрибутах). Зарезервированное слово чувствительно к регистру: True не равно true. Имена атрибутов также чувствительны к регистру.

Приложения всегда получают атрибут в виде строки, поэтому, если приложением используются численные величины, они должны быть преобразованы из строкового представления.

Некоторые атрибуты едины для всех полей. Определение атрибутов необязательно; многие атрибуты имеют значения по умолчанию. Они используются при неопределенном атрибуте. Другие атрибуты специально предназначены для определенного типа полей (например, цвет фона изображения). Попытка присвоить такой атрибут другим полям приведет к сообщению об ошибке.

Можно определить собственные атрибуты с именами, не конфликтующими ни с одним из стандартных имен атрибутов. Имя атрибута, как и ключевое слово, может содержать буквы, цифры или символ подчеркивания (_). Первым символом должна быть буква.

Значения определенных пользователем атрибутов должны соответствовать типам атрибутов полей.

Предопределенные атрибуты

Здесь описаны определенные PDB атрибуты в алфавитном порядке.

Имя атрибута	*Поля*	*Назначение (если определено или true)*
action	Все активные поля	Активное выражение Автолиспа
alignment	Все	Горизонтальное или вертикальное положение в группе
allow_accept	Текстовое поле, клавиша изображения и поле списка	Активизирует клавишу is_default, когда это поле выбрано
acpect_ratio	Изображение, клавиша изображения	Коэффициент коррекции изображения
big_increment	Скользящая шкала	Наибольший шаг перемещения
children_alignment	Ряд, колонка, ряд выбора, ряд в рамке, колонка в рамке, ряд выбора в рамке и колонка выбора в рамке	Выравнивание элемента, подчиненного группе
children_fixed_heigth	Ряд, колонка, ряд выбора, ряд в рамке, колонка в рамке, ряд выбора в рамке и колонка выбора в рамке	Высота элемента, подчиненного группе. Не растет при компоновке
children_fixed_width	Ряд, колонка, ряд выбора, ряд в рамке, колонка в рамке, ряд выбора в рамке и колонка выбора в рамке	Ширина элемента, подчиненного группе. Не растет при компоновке

color	Изображение, клавиша изображения	Цвет фона изображения
edit_limit	Текстовое поле	Максимальное число символов, которое может ввести пользователь
edit_width	Текстовое поле, раскрывающийся список	Ширина редактируемой части поля
fixed_heigth	Все	Высота не растет при компоновке
fixed_width	Все	Ширина не растет при компоновке
heigth	Все	Высота поля
initial_focus	Элемент диалога	Ключ поля с первоначальной подсветкой
is_bold	Надпись	Выводится как полужирный текст
is_cansel	Клавиша	Кнопка активизируется при нажатии комбинации клавиш Ctrl+C.
is_default	Клавиша	Кнопка активизируется при нажатии клавиши "RETURN"
is_enabled	Все активные поля	Поле является изначально доступным
is_tab_stop	Все активные поля	Поле может стать активным при нажатии клавиши табуляции
key	Все активные поля	Имя (ключ), используемое приложением
label	Ряд в рамке, колонка в рамке, ряд выбора в рамке, колонка выбора в рамке, клавиша, примитив диалога, текстовое поле, поле списка, раскрывающийся список, клавиша выбора, надпись и переключатель	Отображаемая метка поля

layout	Скользящая шкала	Скользящая шкала вертикальна или горизонтальна
list	Поле списка, раскрывающийся список	Начальные значения, отображаемые в списке
max_value	Скользящая шкала	Максимальное значение скользящей шкалы
min_value	Скользящая шкала	Минимальное значение скользящей шкалы
mnemonic	Все активные поля	Мнемонический символ для поля
multiple_select	Поле списка	Поле списка позволяет выбрать несколько пунктов
small_increment	Скользящая шкала	Малый шаг перемещения
tabs	Поле списка, раскрывающийся список	Табуляция при отображении списка
value	Текст, активные поля, (за исключением клавиш и клавиш изображения)	Начальное значение поля
width	Все	Ширина поля

Структура DCL-файла

Помимо диалоговых окон DCL-файлы могут определять прототипы или объединения, которые могут включать определения из других DCL-файлов. DCL-файл может состоять из следующих трех частей, которые могут располагаться в любом порядке (некоторые части могут отсутствовать):

- ссылки на другие DCL-файлы. Содержат директивы по включению;
- определения прототипов полей, объединения в колонки и ряды. Это определения полей, на которые можно ссылаться;
- определения диалоговых окон.

Файлы base.dcl и acad.dcl

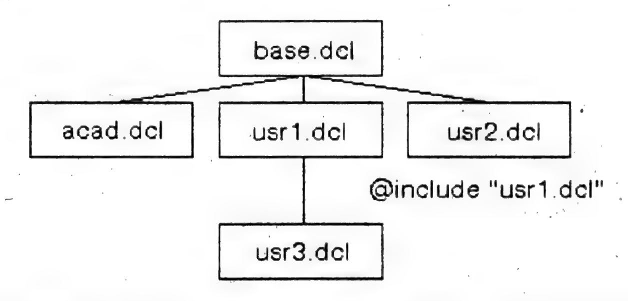
В комплект поставки Автокада включены файлы base.dcl и acad.dcl. Они могут быть использованы в качестве примера при создании DCL-файлов.

Файл base.dcl содержит определения базовых, предварительно определенных полей и типов полей. В нем также содержатся определения прототипов общего использования. Предварительно определенные поля не могут быть переопределены средствами PDB. Не следует модифицировать этот файл! Ошибки в файле base.dcl будут прерывать появление как стандартных диалоговых окон Автокада, так и диалоговых окон приложений пользователя.

Файл acad.dcl содержит определения всех стандартных диалоговых окон, используемых стандартной версией Автокада. Можно отредактировать этот файл, если необходимо изменить появление стандартных диалоговых окон.

Определенные пользователем DCL-файлы

Все определенные пользователем DCL-файлы могут автоматически ссылаться на поля, определенные в файле base.dcl. DCL-файл также может использовать поля, определенные в другом DCL-файле дополнительно к base.dcl, используя имя другого файла в директиве включения (include):



Здесь файлы usr1.dcl и usr2.dcl независимы друг от друга, а файл usr3.dcl использует поля, определенные в файле usr1.dcl.

Директива включения имеет форму

```
@include имя_файла
```

где имя_файла - строка, заключенная в кавычки и содержащая полное имя другого DCL-файла (с расширением dcl).

Создаваемый DCL-файл не может использовать диалоговые окна, определенные в файле acad.dcl. Нельзя использовать директиву @include

“acad.dcl”, хотя для создания подобных диалоговых окон можно вырезать часть из файла acad.dcl и вставить ее в создаваемый файл.

Синтаксис языка DCL

Язык DCL используется для определения новых полей и объединения существующих полей в диалоговые окна. Новые окна создаются определениями полей. Если определение появляется вне диалогового окна, оно является прототипом (определением поля) или объединением (группой с подчиненными элементами), которое можно использовать в диалоговых окнах для ссылок. Каждая ссылка на определение наследует атрибуты исходного поля. Если ссылка дается на прототип, можно изменять значения наследуемых атрибутов или добавлять новые атрибуты; в случае ссылок на объединения атрибуты не могут быть изменены или добавлены.

Если используется несколько экземпляров полей с несколькими общими атрибутами, проще определить прототип, который содержит общие атрибуты. Тогда в каждой ссылке на прототип можно изменять или добавлять новые атрибуты. Вы избавляетесь от необходимости перечислять список всех общих атрибутов при каждой ссылке на поле.

Так как атрибуты наследуются именно таким образом, то при создании диалогового окна проще создать ссылки на поля (особенно ссылки на предварительно определенные поля), чем создавать новые поля.

Определение поля

Форма определения поля

```
name : item1 [ : item2 : item3 ....]{
attribute = value;
}
```

где item - предварительно определенное поле; name - новое поле, которое наследует атрибуты всех определенных полей item; {} - определения атрибутов (добавляют или, если имена атрибутов идентичны, замещают наследуемые определения). Если определение имеет множество родителей, атрибуты имеют приоритет в порядке слева направо. Если более чем одно предварительно определенное поле item определяет один и тот же атрибут, используется первое из них.

Если новое определение не содержит подчиненных элементов, оно является прототипом и ссылки на него могут изменять или дополнять его атрибуты. Если оно является группой (рядом или колонкой) с подчиненными пунктами или вложенными полями, оно является субобъединением.

Имя name поля или прототипа может содержать только буквы, числа или символ подчеркивания (_) и должно начинаться с буквы.

Примеры:

Внутреннее определение клавиши

```
button ; tile {
fixed_heigth = true;
is_tab_stop = true;
        }
```

Файл base.dcl определяет клавишу по умолчанию defoult_button:

```
default_button : button {
is_defoult = true;
        }
```

Поле default_button наследует значения атрибутов fixed_heigth и is_tab_stop поля button. Файл добавляет новый атрибут is_defoult и присваивает ему значение true.

Ссылки на поля

Форма ссылки на поле

```
Name:
```

или

```
: name {
attribute = value;
        }
```

В обоих случаях name - имя предварительно определенного поля. В первом случае все атрибуты, определенные в name, объединены в ссылке. Во втором случае определения атрибутов внутри фигурных скобок добавляют или замещают определения, наследуемые из name. Так как это ссылка на поле, а не на определение, изменения атрибутов относятся только к этому образцу поля. Вторая форма может ссылаться только на прототипы, а не на объединения (группы с подчиненными элементами).

Атрибуты и значения атрибутов

Внутри фигурных скобок определения поля или ссылки можно определить атрибуты и присвоить им значения, используя форму

```
attrubute = value;
```

где attribute - ключевое слово; value - значение, присвоенное атрибуту; = - знак присваивания, ; - завершение присваивания.

Комментарии

Комментариям в DCL-файлах предшествуют две обратные косые черты - //. Все, что появляется между // и концом строки, игнорируется. DCL также позволяет использовать комментарии, принятые в языке Си. Они имеют форму /*, текст комментария - */. Начальные (/*) и конечные (*/) символы могут находиться на разных строках.

Обработка ошибок в DCL

Средства PDB проверяют DCL-файл сразу же после его загрузки. Если при этом обнаруживается синтаксическая ошибка, неправильное использование атрибутов или другая ошибка (например, ошибка определения атрибута key для активного поля), DCL-файл не загружается и в файл acad.dce выводится список ошибок. Если DCl-файл прочитан успешно, Автокад удаляет файл acad.dce.

3.2. Приемы языка DCL

Для правильной компоновки полей в диалоговом окне следует:

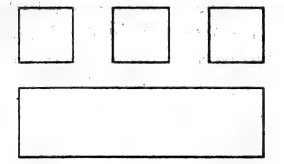
- использовать компоновку по умолчанию и проверить результат: в случае возникновения проблем, похожих на описанные в этом разделе, или специальных случаев, требующих отладки, перейти к следующему шагу;
- попытаться отрегулировать компоновку (изменить значения по умолчанию на уровне групп);
- отрегулировать отдельные поля.

Управление распределением полей в группе

Пусть определен ряд из трех полей, расположенных вдоль другого поля сверху:

```
: column {
: row  {
: compact_tile {
                    }
: compact_tile {
                    }
: compact_tile {
                    }
            }
: large_tile {
            }
}
```

Предполагается, что компоненты поля compact_tile имеют атрибут fixed_width и что поле large_tile шире, чем минимальное пространство, необходимое для расположенного выше ряда полей compact_tile. По умолчанию эта группа выглядит следующим образом:



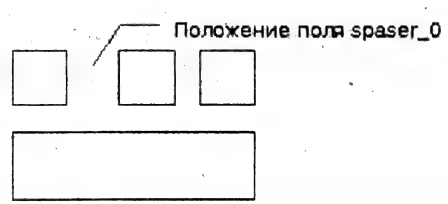
Горизонтальное выравнивание по умолчанию

Левая граница правого поля compact_tile выравнивается по левой границе поля large_tile, а правая граница последнего поля compact_tile ряда выравнивается по правой границе поля large_tile. Поля, расположенные между ними, распределяются равномерно. Аналогично выравнивается группа колонок.

Для изменения распределения по умолчанию следует использовать поля spacer_0 и spacer_1, которые отличаются от определенного в файле base.dcl поля spacer.

Точка вставки наполнения (spaser_0)

Поле spacer_0 в нормальном состоянии не имеет ширины. Оно указывает точку вставки разделителя в группе полей, если группу необходимо растянуть при компоновке. Если группе полей spacer_0 присвоена положительная ширина, то они все будут занимать равную долю пространства.



Распределение, измененное вставкой наполнения

Единичное наполнение (spacer_1)

Поле spacer_1 имеет ширину и высоту, равные единице. Поле используется как наименьший разделитель.

Пример:

Распределение по умолчанию

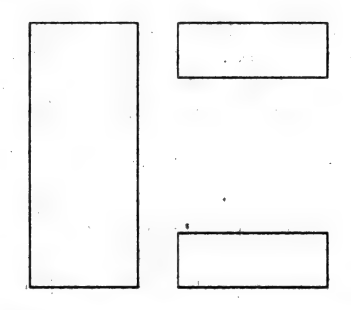
Положение поля spacer_1

Распределение, измененное вставкой единичного наполнения

Нежелательный промежуток между полями в прореженных группах

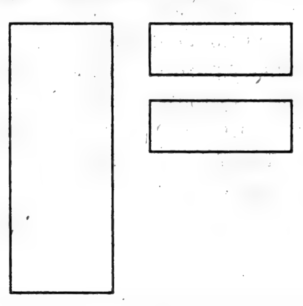
Если смежные колонки существенно различаются по размерам, занимаемым полями, поля меньшего размера в группе будут распределены слишком редко.

Пример:



Поля, распределенные слишком редко

В данном случае колонка будет выглядеть лучше, если присвоить атрибуту fixed_heigth значение true:



Скорректированное прореженное распределение

Нежелательный промежуток справа и внизу диалогового окна

Иногда в диалоговых окнах имеется неиспользованное пространство вдоль правой стороны. Это может быть вызвано полями text, у которых выводимое значение меньше, чем значение атрибута width.

Пример:

Фрагмент диалогового окна режимов создания примитивов Автокада:

```
: text {
key = "l_text";
width = 18;
fixed_width = true;
        }
```

В поле text не отображается ничего (атрибут value не установлен). Приложение может вывести надпись в поле text, используя функцию

```
(set_tile "l_text" "По слою")
```

Так как ширина надписи "По слою" меньше 18 символов, вдоль правой стороны диалогового окна появится пустое пространство. Для того, чтобы ограничить поле text, его можно вставить в ряд в рамке. Иногда для этого может потребоваться перестановка полей в диалоговом окне.

Аналогичная ситуация возникает при использовании поля errtile, отображающем сообщение об ошибке: если сообщение об ошибке не выведено, поле выглядит как дополнительное пространство внизу диалогового окна. В этом случае для вертикального выравнивания рекомендуется вставить дополнительное поле spacer в верхнюю часть диалогового окна.

Пространство вокруг ряда или колонки в рамке

Если атрибут label ряда или колонки в рамке является пробелом (" ") или не задан (""), рамка группы создается, но не содержит текста. Единственный пробел не прерывает рамку. Следует иметь в виду:

- если метка является единственным пробелом, то сжимается пространство по вертикали, которое может занимать метка внутри рамки, но не сжимается пространство по вертикали, которое метка может занимать над рамкой;
- если метка - пустая строка, сжимается пространство по вертикали как внутри рамки, так и вне ее.

Группы текста

Поле текста обычно окружено свободным пространством, как и любой другой тип поля. При сочетании частей текста может возникнуть проблема, например, при выводе сообщения

Текущее время 0800 ч. 37 с.

где значения 0800 и 37 заполняются пользовательским приложением. Чтобы исключить нежелательные свободные промежутки, используются соединенные надписи, построенные из полей text_part. При создании абзаца можно располагать текстовые части (надписи) вертикально и таким образом избежать слишком больших промежутков между строками.

Описанные в этом разделе поля text_part, concatenation и paragraf, являются полями-прототипами, определенными в файле base.dcl.

Текстовые части (text_part)

Текстовая часть - поле надписи, используемое как часть большого куска текста. Граница поля text_part подавляется так, чтобы сочетаться с другими частями text_part в соединении или в поле абзаца.

Соединение (concatenation)

Соединение - строка надписи, созданная из множества соединенных полей text_part. Это может быть полезно при вставке в стандартное сообщение какого-нибудь текста, который может изменяться во время выполнения. Граница вокруг соединения представляется как единое целое.

Абзац (paragraf)

Группа полей text_part или concatenation, расположенных вертикально относительно друг друга. Это позволяет строить абзацы текста как до выполнения, так и во время выполнения. Граница вокруг абзаца представляется как единое целое.

Клавиши выхода из диалогового окна

Файл base.dcl обеспечивает несколько объединений стандартных клавиш для выхода из диалогового окна или его удаления. Эти стандартные объединения следует использовать для сохранения единообразия вида диалоговых окон среди различных приложений. Кроме того, в версии файла base.dcl эти клавиши определены для каждой платформы с учетом ее особенностей, так что использование этих определений гарантирует совместимость с каждой платформой.

Единственная клавиша Да (ОК) - ok_only

Клавиша Да (ОК) аналогична клавише в окне предупреждений. Ключ key клавиши Да имеет значение "accept".

Да (Ok) и Отмена (cancel) - ok_cancel

Комбинация клавиш Да и Отмена (ОК and Cancel) - стандартная комбинация для диалоговых окон, в которых могут вноситься изменения в данные. Ключ key клавиши Отмена (Cancel) имеет значение "cancel".

Да (Ok), Отмена (Cancel) и Помощь (Help) - ok_cancel_help

Группа ok_cancel скомбинирована со стандартной клавишей Помощь (Help). Ключ key клавиши Помощь имеет значение "help".

Да (Ok), Отмена (Cancel), Помощь (Help) и Информация (Info) - ok_cancel_help_info

Все описанные выше клавиши плюс информационная клавиша для вывода дополнительной информации, которая может отображать имя приложения, товарный знак, номер версии, информацию о получении технического сопровождения и т. д. Ключ key клавиши Инфо... имеет значение "info".

Изменение надписи клавиши выхода

Иногда может возникнуть необходимость изменить надпись клавиш выхода, например для создания диалогового окна, которое может удалить данные, используя клавишу Удаление вместо клавиши Да. Для этого следует использовать прототип retirement_button:

```
destroy_button : retirement_button {
                label    = "Удаление";
                key      = "destroy";
                mnemonic = "y";
        }
```

После определения измененной клавиши выхода следует встроить ее в объединение, которое соответствует по виду и действию стандартным группам, описанным в предыдущем разделе. Например, здесь определено поле ok_cancel_help:

```
ok_cancel_help : column {
: row {
fixed_width = true;
```

```
alignment =centered;
ok_button:
: spacsr {width =2;}
cancel_button:
: spacer {width =2;}
help_button;
                    }
        }
```

Описание нового объединения с замещенной клавишей ok_button на новую:

```
        }
```

В стандартном объединении клавиша Да предлагается по умолчанию, но атрибут is_default не добавляется к описанию destroy_button. В подобных ситуациях, если диалоговое окно выполняет действия по удалению, следует сделать клавишу Отмена клавишей по умолчанию. В этом случае она действует и как клавиша по умолчанию, и как клавиша отмены опасной операции:

```
destroy_cancel_help : column {
:row {
fixed_width = true;
alignment = centered;
destroy_button :
: spacer {width = 2;}
: cancel_button {is_default = true;}
spacer {width = 2;}
help_button;
                          }
                  }
```

Перед cancel_button ставится двоеточие. Поскольку атрибут изменен, следует использовать исходное описание клавиши Отмена как прототип. Если клавиша отмены и клавиша по умолчанию совпадают (обе имеют атрибуты is_default и is_cancel, установленные в true, как в этом примере) и не присвоено действие с вызовом функции done_dialog любой другой клавише, то ни одна клавиша не сможет закрыть диалоговое окно, и его действие всегда будет отменяться.

Поле ошибки - errtile

Поле надписи, которое, как правило, появляется внизу диалогового окна. По умолчанию оно пусто, но приложение может отобразить в нем сообщение, присвоив значение полю с ключом 'error".

Да (Ok), Отмена (Cancel), Помощь (Help) и поле ошибки - ok_cancel_help_errtile

Объединение, определенное в файле base.dcl, является колонкой, в которой клавиши выхода ok_cancel_help располагаются над стандартным полем errtile. Этим достигается удобное определение одновременно клавиш выхода и поля ошибки.

3.3. Управление диалоговыми окнами

Описание диалогового окна на языке DCL является статической картиной формы окна. Все действия во время диалога и после закрытия окна определяются управляющей программой вызова диалогового окна и обработки результатов диалога. В данном пособии мы рассматриваем управление диалоговыми окнами только через Автолисп. Сначала опишем применяемые для этого функции Автолиспа, затем посмотрим, как их использовать в управляющих программах. Такие управляющие программы могут содержать не только строки управления диалогом, но и выполнять определенные прикладные задачи пользователя. Однако более целесообразно функции управления диалоговыми окнами оформлять в виде отдельных подпрограмм, вызываемых из прикладной задачи. В этом случае облегчается режим отладки диалога и появляется возможность использования таких функций в других прикладных программах.

3.4. Функции Автолиспа для диалоговых окон

Эти функции обращаются к соответствующему DCL-файлу при отображении диалогового окна на экране.

Открытие и закрытие DCL-файлов

(load-dialog имя-файла)

Загружает указанный DCL-файл. Возвращает целое значение (dcl_id), используемое в последующих вызовах функций new_dialog и unload_dialog. В имя файла не следует вставлять расширение.

(unload-dialog dcl_id)

Выгружает указанный DCL_файл. Всегда возвращает nil.

Открытие и закрытие диалоговых окон

(new_dialog имя-окна dcl_id [[действие] точка])

Начинает управление диалоговым окном, выводит его на экран и может определить действие по умолчанию. Аргумент имя-окна является строкой и определяет диалоговое окно, а аргумент dcl_id определяет DCL-файл (это значение определяется вызовом функции load_dialog). Приложение должно вызывать функцию new_dialog перед вызовом start_dialog. Все установки, такие, как присвоение полям значений, создание изображений или списков для полей списка и связывание действий с полями (через вызовы функции action_tile), должны происходить после вызова функции new_dialog и перед вызовом start_dialog. Действие по умолчанию выполняется после указания пользователем активного поля, которое не имеет присвоенного функцией action_tile или определенного в DCL-файле действия или функции вызова с возвратом. При успешном завершении функции (new_dialog) возвращает t, в противном случае - nil. Аргумент действие, который должен быть задан, если определен аргумент точка, является строкой, содержащей выражение Автолиспа для использования как действия по умолчанию. Если аргумент действие определять не нужно, следует присвоить ему пустую строку (“ ”). Если присутствует аргумент точка, то он является списком (X Y) точки расположения диалогового окна на экране. Точка обычно определяет положение левого верхнего угла диалогового окна. Поэтому не следует определять положение диалогового окна при его первой инициализации. Координаты положения окна следует задавать после вызова функции (done_dialog) для повторного открытия окна. Это позволяет открывать окна заново, где они были закрыты. Если точка определена как ‘(-1 -1), диалоговое окно откроется в положении по умолчанию (в центре графического экрана Автокада).

Всегда следует проверять возвращаемое функцией new_dialog значение. Вызов функции start_dialog, если функция new_dialog вернула значение ошибки, может привести к непредсказуемым последствиям.

(start_dialog)

Начинает диалог в диалоговом окне. Диалоговое окно должно быть предварительно инициализировано предшествующим вызовом функции new_dialog. Оно остается активным, пока выражение действия или функция вызова с возвратом не вызовут функцию done_diflog; обычно вызов done_dialog связан с полем, имеющим ключ “accept” (обычно клавиша “Да” (“OK”)) или “Cancel” (обычно клавиша “Отмена” (“Cancel”)).

Функция (start_dialog) не имеет аргументов. Она возвращает аргумент ***состояние***, переданный в функцию (done_dialog). Значением по умолчанию является 1, если пользователь указал клавишу Да; 0, если пользователь указал клавишу “Отмена”; -1, если окна были закрыты вызовом функции (term_dialog). Но если функция (done_dialog) установила

состояние в значение большее 1, функция (start_dialog) вернет значение, смысл которого определяется приложением.

(done_dialog [состояние])

Завершает диалог в диалоговом окне и убирает его с экрана. Функция должна вызываться из выражения действия или из функции вызова с возвратом. Эта функция также возвращает текущее положение (X Y) диалогового окна. Если функция с возвратом определена для клавиш с ключом "accept" или "done", функция вызова с возвратом должна определенно вызывать функцию done_dialog. Если этого не сделать, пользователь может не выйти из диалогового окна. Если с этими клавишами не связана функция вызова с возвратом и используются стандартные клавиши выхода, Автокад обрабатывает их автоматически. Кроме того, точное действие Автолиспа для клавиши "accept" должно определить действие как 1 (или как определенное приложением значение), иначе функция (start_dialog) вернет значение по умолчанию 0, что аналогично отмене диалогового окна.

Аргумент ***состояние*** не является обязательным. Если он определен, то должен быть положительным целым, который функция (start_dialog) возвращает как 1 для Да и как 0 для Отмена. Функция (done_dialog) возвращает список координат точки (X Y), определяющей положение диалогового окна, которое покинул пользователь. Этот список можно использовать в последующем вызове функции (new_dialog) для повторного открытия диалогового окна в месте, выбранном пользователем.

(term_dialog)

Завершает диалог во всех активных в настоящий момент диалоговых окнах, как в случае прерывания пользователем. Если при открытых DCL-файлах выполнение приложения прерывается, Автокад автоматически вызывает функцию (term_dialog). Данная функция используется в основном для закрытия вложенных диалоговых окон. Функция (term_dialog) всегда возвращает nil.

Инициализация выражений действия и функций вызова с возвратом

(action_tile ключ выражение_действия)

Присваивает действие, которое будет выполняться после выбора пользователем определенного поля. Аргумент ключ представляет имя поля, которое будет вызывать действие. Действие, присвоенное функцией action_tile, замещает действие по умолчанию диалогового окна или атрибут поля action, если они определены в DCL-файле.

Аргументы ключ и выражение_действия являются строками. Аргумент выражение_действия вычисляется после выбора пользователем поля. Выражение может обращаться к текущему значению поля (атрибут поля value), определенному как $value, к его имени, определенному как

Skey, к данным, установленным функцией (client_data_tile), определенным как Sdata, к коду причины вызова, определенному как Sreason, к координатам изображения, если это поле изображения, определенным как Sx и Sy.

Обработка полей и атрибутов

(mode_tile ключ режим)

Устанавливает режим для данного поля. Аргумент ключ определяет поле. Аргумент режим является целым числом, его значения даются в таблице:

Значение аргумента	*Результат*
0	Поле включено (доступно)
1	Поле отключено (недоступно)
2	Выбрать поле
3	Подсветить содержимое текстового поля
4	Включить/отключить подсветку изображения

*Для аргумента **режим** можно определить только одно целое значение, Аргумент **ключ** является строкой.*

(get_attr ключ атрибут)

Запрашивает значение DCL указанного атрибута. Аргумент ключ определяет поле, аргумент атрибут - имя атрибута в DCL-описании поля. Значение возвращается как первоначально установленное в описании поля; оно не отражает изменений состояния поля, которые могут произойти после ввода пользователя или вызова функции set_tile.

Функция (get_attr) возвращает значение атрибута как строку. Аргументы ключ и атрибут являются строками.

(get_tile ключ)

Запрашивает действующее значение указанного поля.

Аргумент ключ определяет поле. Функция (get_tile) возвращает значение поля как строку. Аргумент ключ является строкой.

(set_tile ключ значение)

Устанавливает действующее значение для указанного поля. Аргумент ключ определяет поле, аргумент значение - присваиваемое значение. Все аргументы являются строками.

Задание полей списков и раскрывающихся списков

(start_list ключ [операция] индекс])

Запускает обработку указанного поля списка или раскрывающегося списка, определенного аргументом ключ. Аргумент операция является целым числом со следующими возможными значениями:

Аргумент	*Результат*
1	Изменить выбранное содержимое списка
2	Добавить новый пункт в список
3	Удалить старый список и создать новый (по умолчанию)

Аргумент индекс игнорируется, если функция start_dialog не вызывается с кодом изменения 1, тогда индекс определяет пункт списка для изменения последующим вызовом функции add_list. Значения индекс начинаются с нуля. Аргументы операция и индекс не обязательны. Если не определен аргумент операция, он по умолчанию устанавливается в 3. Если не определена операция и не определен индекс, индекс по умолчанию устанавливается в 0. Аргумент ключ является строкой.

(add_list элемент)

Добавляет заданную строку в текущий список или замещает пункт списка на элемент. Аргумент элемент является строкой. Он завершает обработку текущего списка.

(end_list)

Создание изображений

(dinx_tile ключ)
(diny_tile ключ)

Функции возвращают размеры поля в единицах диалогового окна, которые используются функциями vector_image, fill_image и slide_image, требующими задания абсолютных координат. Аргумент ключ определяет поле. Координаты возвращаются как максимально разрешенные в поле. Так как координаты отсчитываются от нуля, функции возвращают значения на единицу меньше, чем размеры по X- или Y-направлению. Функция dimx_tile возвращает ширину поля, функция dimy_tile - его высоту. Для обеих функций аргумент ключ является строкой.

(start_image ключ)

Запускает процесс создания указанного изображения в поле, определенном аргументом ключ. Аргумент ключ является строкой.

(vector_image x1 y1 x2 y2 цвет)

Рисует вектор на текущем активном изображении (открытом функцией start_image) из точки (x1 y1) в точку (x2 y2). Параметр цвет определяет номер цвета Автокада или один из логических цветов:

Номер цвета	*Мнемоника ADI*	*Значение*
-2	BGLCOLOR	Цвет фона графического экрана Автокада
-15	DBGCOLOR	Цвет фона диалогового окна
-16	DFGCOLOR	Цвет диалогового окна (для текста)
-18	LINECOLOR	Цвет линии диалогового окна

Начало (0 0) находится в левом верхнем углу изображения. Можно получить координаты правого нижнего угла изображения, вызвав функцию размеров (dimx_tile).

(fill_image x1 y1 x2 y2 цвет)

Рисует закрашенный прямоугольник на активном изображении. Аргументы аналогичны функции vector_image.

(slide_image x1 y1 x2 y2 имя_слайда)

Отображает слайд Автокада на текущем активном изображении. Слайд может быть отдельным слайдом (*. sld) или находиться в составе библиотеки слайдов (*.slb). При вызове из библиотеки сначала указывается имя библиотеки, затем в круглых скобках имя слайда. Первый угол слайда, его точка вставки, определяется как (x1 y1), второй угол - (x2 y2). Начало (0 0) находится в левом верхнем углу изображения. Завершает создание текущего изображения.

(end_image)

Данные, связанные с программным приложением

(client_data_tile ключ клиент-данные)

Связывает управляемые приложением данные с полем, определенным аргументом ключ. Аргумент ключ является строкой. Определенные приложением данные задаются аргументом клиент-данные и также являются строкой. Выражение действия может ссылаться на строку, определенную как $data.

3.5. Схема вызовов функций управления

Рассмотрим пример вызова и управления простого диалогового окна

```
hello : dialog {
label = "Пример диалогового окна";
: text {
label = "Всем привет";
                    }
ok_only;
        }
```

Данное описание находится в файле hello.dcl. Управляющая появлением этого окна функция Автолиспа может иметь следующий вид:

```
(defun showalert ( / dcl_id)
        (setq dcl_id (load_gialog "hello.dcl"))    ;Загрузить DCL-файл
        (if (not (new_dialog "hello" dcl_id))      ;Инициализировать диалог
        (exit))                                    ;(выйти, если не работает)
        (action_tile                               ;Связать выражение действия
        "accept"                                   ;с ключом клавиши Да
        "(done_dialog)")                           ;Закончить диалог, если
                                                   ;нажата клавиша Да
        (start_dialog)                             ;Вывести диалоговое окно
        (unload_dialog dcl_id)                     ;Выгрузить DCL-файл
)
```

После вызова функции (start_dialog) диалоговое окно становится активным, пока пользователь не подсветит поле (обычно клавишу), которое связано с вызовом выражения (done_dialog). Вызов функции (action_tile) устанавливает связь между полем (в этом примере с клавишей Да, ключ которой имеет значение "accept") и выражением действия. Именно из-за этого вызов функции (done_dialog) появляется внутри вызова функции (action_tile) и перед вызовом функции (start_dialog). По сути, все действия до функции (start_dialog) являются настроечными. Более сложные окна потребуют большего числа настроек и дополнительных операторов между функциями (start_dialog) и (unload_dialog), но последовательность вызовов будет такой же.

Показанный пример демонстрирует обычную последовательность вызовов функций:

1. Загрузка DCL-файла функцией (load_dialog).

2. Вызов функции (new_dialog) для вывода отдельного диалогового окна на графический экран Автокада. Важно проверять возвращаемое функцией (new_dialog) значение. Вызов функции (start_dialog), если функция (new_dialog) возвратила ошибку, может привести к непредсказуемым результатам.

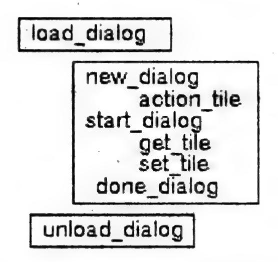
3. Инициализация диалогового окна и установка значений полей, списков и изображений, если это необходимо. На этом этапе обычно вызываются функции (set_tile) и (mode_tile) для установки состояния полей и их значений; (start_list), (add_list), (end_list) для полей списков; (start_image), (vector_image), (fill_image), (slide_image), (end_image) для изображений. На этом этапе обычно вызывается функция (action_tile) для установки выражения действия. Здесь можно вызвать и функцию (action_tile) для связи определенных приложением данных с диалоговым окном и его компонентами.

4. Вызов функции (start_dialog) передает управление диалоговому окну, в которое пользователь может вводить данные.

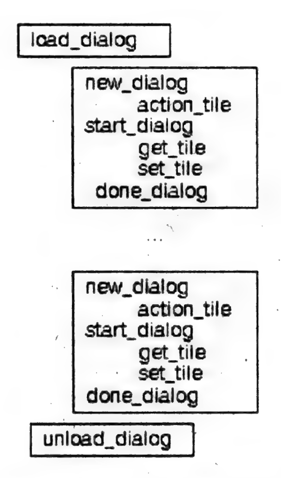
5. Процесс ввода данных пользователем (вызовы с возвратом). На этом этапе используются функции (get_tile), (get_attr), (set_tile), (mode_tile).

6. Указание пользователем клавиши выхода вызовет функцию (done_dialig), которая, в свою очередь, возвращает функции (start_dialog). Далее следует выгрузка DCL-файла, обеспечиваемая функцией (unload_dialog).

Показанную последовательность действий можно представить следующей схемой:



На схеме показано одновременное управление только одним окном и одним DCL-файлом. Управление несколькими окнами аналогично: следует размножить средний большой прямоугольник нужное число раз:



Схему можно использовать для загрузки нескольких DCL-файлов.

Функции, запрещенные во время действия диалогового окна

При активном диалоговом окне (функция (start_dialog) вызвана) нельзя вызывать некоторые функции Автолиспа, которые изменяют экран, который не должен изменяться, пока на нем изображается диалоговое окно, или требуют ввода пользователя, не имеющего отношения к диалоговому окну.

Если необходимо ввести данные в режиме графического экрана (например, выбрать точку или примитив), необходимо временно закрыть диалоговое окно, вызвав функцию (done_dialog), - графический экран станет доступен. После выполнения выбора производится повторный вызов окна.

Перечень функций Автокада, запрещенных к вызову при открытом диалоговом окне:

1. Запросы и команды Автокада: (command), (osnap).

2. Функции ввода пользователем: (getint), (getreal), (getstring), (getpoint), (getcorner), (getdist), (getangle), (getorient), (getkword).

3. Функции управления экраном: (prompt)[1], (menucmd), (redraw), (graphscr), (textscr), (textpage).

4. Графические функции низкого уровня: (grclear), (grdraw), (grread), (grtext), (grvacs).

5. Функция набора выбора (ssget)[2].

6. Функции управления примитивами: (entmod), (entmake), (entdel), (entsel), (nentsel), (entupd).

Выражения действий

Для определения последствий выбора определенного поля диалогового окна следует связать выражение Автолиспа с этим полем посредством вызова функции (action_tile). Внутри действия часто бывает необходимо получить доступ к атрибутам DCL-файла. Это обеспечивают функции (get_attr) и (get_tile): функция (get_attr) выдает значения, сохраненные в DCL-файле, а (get_tile) выдает текущие значения. Значения, связанные с выбранным полем, получаются автоматически.

Выражение действия Автолиспа получает доступ к переменным, которые описаны ниже в таблице и описывают выбранное поле и его состояние во время действия. Имена переменных зарезервированы, и их значения можно только считать (они имеют смысл только тогда, когда доступны выражению действия).

---

[1] Функции вывода текста (print) и (princ) полезно использовать при отладке диалоговых окон, но не в окончательной версии. Если диалоговое окно перекрывает зону подсказок, эти функции выводят текст поверх диалогового окна.

[2] Запрещены только интерактивные функции (ssget).

Переменная	*Назначение*	*Примечание*
$key	Атрибут key выбранного поля	Применима ко всем действиям
$value	Строка текущего значения поля, похожая на строку окна редактирования, или "0", или "1" для переключателя	Применима ко всем действиям. Если поле является полем списка (или раскрывающегося списка) и не содержит ни одного пункта, переменная $value будет иметь значение nil
$data	Данные, установленные сразу после вызова (new_dialog) через вызов (client_data_tile) и управляемые приложением	Применима ко всем действиям. Не имеет значения до тех пор, пока приложение не установит его вызовом функции (client_data_tile)
$reason	Код причины, сообщающий, какой выбор пользователя вызвал действие. Используется в полях edit_box, list_box, image_button и slider	Определяет причину действия. Ее значение определено для любого вида действия, но его следует проверять, если действие связано с полями edit_box, list_box, image_button и slider
$x	Координата X выбора image_button.	Представляют координаты X и Y точки, в которой пользователь выбрал поле image_button. Для других целей эти переменные не имеют значения. Координата X находится в интервале, возвращаемом функцией (dimx_tile), координата Y - в интервале, возвращаемом функцией (dimy_tile).
$y	Координата Y выбора image_button.	

Примеры:

Если текстовое поле определено как edit1, выражение действия в следующем вызове функции (action_tile) вычисляется после того, как пользователь покинет текстовое поле:

```
(action_tile "edit1" "(setq ns $value)")
```

В этот момент $value содержит строку, введенную пользователем, и она сохранится в переменной ns.

В следующем примере сохраняется имя выбранного поля, и в случае необходимости программа может ссылаться на него.

```
(action_tile "edit1" "(setq newtile $key)")
```

Переменная newtile устанавливается в значение атрибута key выбранного поля, т. е. в "edit1". Переменная $key часто используется внутри функций, которые служат для выполнения действий, присвоенных нескольким различным полям.

Вложенные диалоговые окна

Создавая вложенные диалоговые окна, ими можно управлять с помощью функций (new_dialog) и (start_dialog), вызывая их выражение действия. Например, функция (showalert) отобразила бы диалоговое окно "Всем привет" при выборе пользователем клавиши button_1 какого-либо нового диалога, если вставить выражение

```
(action_tile "button_1" "(showalert)").
```

Для возвращения в предыдущее окно необходимо покинуть текущее.

Слишком большое число вложений усложняет работу. Автокад ограничивает число вложений восемью. Рекомендуется использовать около трех-четырех вложенных окон.

Временное закрытие диалоговых окон

При необходимости выхода из диалогового окна для интерактивного выбора на графическом экране следует временно закрыть диалоговое окно и после выполнения действия на графическом экране вернуться в диалоговое окно с полным его восстановлением. Такая процедура выполняется с помощью функции (done_dialog) с аргументом status.

Пример:

В программе bmake.lsp используется клавиша Указание точки<, которая временно закрывает диалоговое окно для предоставления возможности ввода точки на графическом экране. При указании этой клавиши происходит закрытие диалогового окна с кодом состояния 4:

```
(action_tile "pick_pt" "(done_dialog 4)")
```

После возврата из функции (start_dialog) программа проверяет возвращаемый код состояния и при необходимости изменяет координаты точки:

```
(setq what_next (start_dialog))
(cond ((=  what_next 4)
(progn
(setq pick_pt (getpoint "Базовая точка вставки: "))
(setq x_pt (rtos (car (pick_pt) 2 4))
(setq y_pt (rtos (cadr (pick_pt) 2 4))
(setq z_pt (rtos (caddr (pick_pt) 2 4))
                              ))
                  )
```

Тело основной функции заключено в цикл с вызовом функции (start_dialog) до тех пор, пока пользователь не нажмет на клавишу Да или Отмена:

```
(defun c:mblock ...
...
(while (< 2 what_next)
...
(setq what_next (start_dialog)
(cond
...
                    )
          ))
```

Если диалоговое окно вложено, процедура усложняется, по существу оставаясь прежней:

```
(defun c:mdialog (/what_next what_next1)
(setq what_next 5)
(if (< (setq dcl_id (load_dialog "maindlg.dcl")) 0) (exit))
(while (< 1 what_next)
(new_dialog "maindlg" dcl_id)
(action_tile "x" "(subdlg)")
(cond
((= what_next1 3)
(subdlg)
(if (/= 3 what_next1)
(setq what_next (start_dialog))
                                    )
                              )
(T (setq what_next (start_dialog)))
                            )
(cond
((= 2 what_next)
(setq xx (getpoint "\nУкажите точку: "))
                              )
                            )
                    )
)
(defun subdlg()
(new_dialog "subdlg" dcl_id)
(action_tile "hide_all" "(done_dialog 3)")
(setq what_next (start_dialog))
(if (= 3 what_next1)
(done_dialog 2)
          )
)
```

Обработка полей

Как управлять полями при инициализации и в процессе ведения диалога? При изменении значения одного поля со связанными с ним полями должны происходить изменения.

Инициализация режимов и значений

При инициализации диалогового окна определенное поле может быть первоначально включено или отключено и подсвечено или не подсвечено. Эти операции выполняются с помощью вызова функции (mode_tile). Установить значение поля можно с помощью функции (set_tile). Во время инициализации делаются установки полей списка и создаются изображения.

Пример:

Показанный набор функций выполняет установку значения по умолчанию для текстового поля и подсвечивает его:

```
(setq name_str "Коленчатый вал")
(set_tile "lastname" name_str)
(mode_tile "lastname" 2)
```

Дополнительный вызов функции (mode_tile) может подсветить содержимое текстового поля:

```
(mode_tile "lastname" 3)
```

Перед тем, как отключить подсвеченное поле, необходимо с помощью вызова функции (mode_tile) подсветить другое поле. Ведь отключенное подсвеченное поле бессмысленно, что в некоторых случаях может привести к ошибке.

Пример "самоотключения" поля - последовательность "страниц" диалоговых окон, которые перелистываются указанием клавиши "Следующий" или "Предыдущий". Когда указывается клавиша "Следующий" на предпоследней "странице", после перехода эта клавиша отключается. То же самое происходит с клавишей "Предыдущий" при переходе на первую "страницу". В обоих случаях использованная клавиша отключается и подсвечивается другое поле.

Предположим, что переключатель с именем "group_on" управляет группой "group": когда переключатель отключен, поля группы недоступны и не могут быть изменены. В этом случае для переключателя следует определить следующее действие:

```
(action_tile "group_on" "(mode_tile \"group\" (- 1 (atoi $value)))")
```

Вычитание и вызов функции (atoi) в выражении действия устанавливают значение аргумента mode функции (mode_tile). Так как значение переключателя равно нулю, когда он отключен, и единице, когда он

включен, вычитание изменяет значение аргумента mode на противоположное и управляет доступностью группы.

Установка полей списков и раскрывающихся списков

При установке значений для поля списка или раскрывающегося списка необходимо выполнить следующую последовательность операций:

(start_list), (add_list) и (end_list)

Список можно изменить после его создания. Функция (mapcar) полезна для перевода списка из Автолиспа в отображаемое поле списка:

```
(start_list "selections" операция)
(mapcar 'add_list newnames)
(end_list)
```

Обработка значений списков

При обработке списков значение поля list_box может содержать начальные пробелы, особенно если выбрано несколько пунктов, так что не следует его проверять с помощью функций сравнения строк. Сначала следует преобразовать значение к целому, используя функцию (atoi) или (read).

Пример:

Предполагается, что список "justone" позволяет выбрать только один пункт одновременно. Фрагмент кода проверяет, выбран ли третий пункт списка. (Поскольку функция (atoi) возвращает 0, если строка пустая или когда в ней содержится "0", сначала следует проверить, является ли строка пустой.)

```
(setq index (get_tile "justone"))
(cond
((/= index "")
(= 2 (atoi index)) ; это третий пункт
...
          )
       )
```

Значение раскрывающегося списка никогда не начинается с символа пробела, так что преобразование в этом случае выполнять не обязательно. Раскрывающиеся списки не позволяют выполнять множественный выбор. Если в поле списка можно выбрать одновременно несколько пунктов, программа должна не только выполнить преобразование, но и перебрать все множество значений в строке.

Пример:

Функция (mk_list) возвращает список, содержащий только выбранные пользователем поля из исходного списка displist. Переменная displist определена как глобальная. Предполагается, что функция (mk_list) вызывается с текущим значением $value поля списка.

```
(defun mk_list (readlist / count item retlist)
(setq count 1)
(while (setq item (read readlist))
(setq retlist (cons (nth item displist) retlist))
(while (and (/= " " (substr readlist count 1))
(while (and (/= "" (substr readlist count 1))
(/= " " (substr readlist count 1)))
(setq count (1+ count))
                                              )
(setq readlist (substr readlist count))
                                    )
(reverse retlist)
          )
```

Создание изображений

Последовательность вызовов для создания поля изображения и клавиши изображения аналогична последовательности обработки списка. Функция (start_image) начинает создание изображения, функция (end_image) заканчивает его. Однако определение результата рисования осуществляется путем использования нескольких разных функций, а не путем задания разных аргументов:

- (vector_image) - рисует вектор (одна прямая линия) на текущем изображении.
- (fill_image) - рисует закрашенный прямоугольник.
- (slide_image) - отображает слайд Автокада.

Эти функции требуют определения абсолютных координат. Чтобы задать их корректно, следует знать размеры поля изображения или клавиши изображения. Так как размеры полей определяются во время их компоновки, функции PDB обеспечивают получение ширины и высоты поля: (dimx_tile) и (dimy_tile). Начало поля (0 0) всегда находится в верхнем левом углу.

Примеры:

Предположим, что нужно закрасить красным цветом поле изображения "cur_color":

```
(setq width (dimx_tile "cur_color")
height (dimy_tile "cur_color"))
(start_image "cur_color")
(fill_image "cur_color")
```

```
(fill_image 0 0 width height 1) ; 1 -  красный цвет Автокада
(end_image)
```

Далее нарисуем вокруг поля границу красного цвета, а не закрашенный прямоугольник (векторы рисуются против часовой стрелки):

```
(setq width (dimx_tile "border")
height (dimy_tile "border"))
(start_image "border")
(vector_image 0 0 0 height 1)
(vector_image 0 height width height 1)
(vector_image width height width 1)
(vector_image width 0 0 0 1)
(end_image)
```

Функции отрисовки должны вызываться вместе. Изобразим закрашенное изображение и нарисуем на нем вертикальную линию:

```
(setq width (dimx_tile "striple")
height (dimy_tile "striple"))
(start_image "striple")
(fill_image 0 0 width height 3); 3 - зеленый цвет Автокада
(setq x(/ width 2.0)
(vector_image x 0 height 4) ; 4 - фиолетовый цвет Автокада
(end_image)
```

Отображаемые с помощью функции (slide_image) слайды могут быть отдельными слайд-файлами (.sld) или частью файла библиотеки слайдов (.slb). Расширение указывать не нужно. Просто слайд именуется по имени, слайд из библиотеки именуется как в меню: сначала имя библиотеки, затем имя слайда в скобках. Функция (slide_image) ищет слайд или библиотеку слайдов в соответствии с текущим путем поиска библиотек Автокада.

Предположим, что необходимо отобразить слайд tppview.sld:

```
(setq x (dimx_tile "view")
y (dimy_tile "view"))
(start_image "view")
(slide_image 0 x y "topview")
(end_image)
```

Векторы слайда часто рисуются белым цветом, который является цветом фона изображения по умолчанию. Если слайд после вывода не виден, надо изменить атрибут color на значение graphic_background.

Ввод клавиши изображения

Клавишу изображения можно обрабатывать как и обычную клавишу, т. е. при ее указании вызывается соответствующее действие. Однако имеется возможность определения клавиши так, что действие будет зависеть от места указания в зоне клавиши. Для этого действие клавиши

должно получить координаты точки указания. Координаты задаются в диапазоне размера изображения и возвращаются функциями определения размеров.

Пример:

Допустим, что клавиша изображения имеет две прямоугольные области разного цвета. Требуется определить, какую из областей указал пользователь. Если клавиша разделена горизонтально, следует проверять только один размер:

```
(action_tile "image_set" "(pick_shade" $key $value $y)")
...
...
(defun pick_shade (key val y)
(setq threshold (/ dimy_tle key) 2))
(if (y threshold)
(setq result "Светлый")
(setq result "Темный"))
)
```

Обработка групп выбора

Кнопки выбора появляются в группах выбора. Значение каждой отдельной кнопки может быть равно единице, если она включена, или нулю, если она отключена. Значением группы кнопок выбора является ключ, выбранной в данный момент кнопки. Проводится проверка включения только одной кнопки. Приложение может присвоить действие каждой отдельной кнопке выбора, но более приемлемым является присвоение действия группе кнопок выбора как единому целому и использование присылаемого ключа для определения, какая из кнопок выбрана в данный момент.

Пример:

Допустим, что группа кнопок выбора определяет вид трехмерного объекта, который будет отображен после закрытия диалогового окна. При этом группа состоит из четырех кнопок (их может быть и больше):

```
(action_tile "view_set" "(pick $value)")
...
...
(defun pick_view (which)
        ((cond ((= which "front") (setq show_which 0))
        ((cond ((= which "top") (setq show_which 1))
        ((cond ((= which "left") (setq show_which 2))
        ((cond ((= which "right") (setq show_which 3))
        )
)
```

Этот пример показывает, что все кнопки выбора связаны с одной переменной, которая может иметь несколько значений. Кроме того, кнопки выбора могут выполнять еще и некоторые дополнительные действия, например отменять сделанный в диалоговом окне выбор.

Обработка скользящих шкал

Действие скользящей шкалы проверяет код причины. В качестве примера покажем базовую схему функции управления скользящей шкалой. Она вызывается из связанного с ней выражения действия. Поле slider_info используется этой функцией для вывода в десятичной форме текущего значения состояния шкалы и является текстовым полем, которое дает пользователю возможность управлять состоянием скользящей шкалы, непосредственно вводя значения. Если пользователь ввел значение в поле slider_info, функция текстового поля изменит значение состояния скользящей шкалы:

```
(action_tile "my_slider" "(slider_action $value $reason)")
(action_tile "my_info" "(ebox_action $value $reason)")
...
...
(defun slider_action (val why)
        (if (or (= why 2) (= why 1))
          (set_tile "slider_info" val)
          )
)
(defun ebox_action (val why)
        (if (or (= why 2) (= why 1))
          (set_tile "myslider" val)
          )
)
```

Обработка текстовых полей

Действия, управляющие текстовыми полями, аналогичны функциям обработки скользящих шкал, но при этом значение текстового поля всегда видимо и нет необходимости выполнять действия, связанные с промежуточными результатами.

Пример проверки значения без вывода его заново:

```
(actio_tile "myeditbox "(edit_action $value $reason)
```

```
...
...
(defun edit_action (val why)
        (if (or (= why 2) (= why 4))
          (set_tile "myeditbox" val)
          )
        )
)
```

Определяемые приложением данные

Функция client_data-tile присваивает полю определяемые приложением значения переменных. Эти данные доступны аналогично переменной Автолиспа Sdata. Данные приложения имеют смысл только во время исполнения приложения. Использование данных приложения сравнимо с использованием определенных пользователем атрибутов. Основное различие состоит в том, что атрибуты пользователя предназначены только для чтения, в то время как данные приложения могут изменяться во время выполнения программы и пользователь может проверить определенный им атрибут в DCL-файле приложения, а данные приложения для пользователя недоступны. В Автолиспе данные приложения определены текстовыми строками.

Пример:

Так как программа должна поддерживать список, выводимый в поле списка (или в раскрывающемся списке), данные приложения могут быть использованы для обработки этой информации. В Автолиспе следующая модификация определения функции (mk_list) возвращает список как аргумент:

```
(defun mk_list (readlist displist)
(setq count 1)
(while (setq item (read readlist))
(setq retlist (cons (nth item displist) retlist))
(while (and (/= " " (substr readlist count 1))
(while (and (/= "" (substr readlist count 1))
(/= " " (substr readlist count 1)))
(setq count (1+ count))
                    )
(setq readlist (substr readlist count))
                    )
(reverse retlist)
          )
```

Это обстоятельство устраняет необходимость в переменной глобального списка. Следующие вызовы в основной части программы обработки диалоговых окон связывают короткий список с полем: вызывают функцию (client_data_ tile), затем передают список в функцию (mk_list) посредством выражения действия:

```
(client_data_tile "colorsyslist"
          "Red-Green-Blue Cyan-Magenta-Yellow Hue-Saturation-Value")
(action_tile "colorsyslist" "(setq usrchoice (mk_list $value $data))")
```

3.6. Примеры разработки диалоговых окон

Рассмотрим использование диалоговых окон в Лисп-программах. Примером такого использования может служить программа построения поверхности, описанной координатами точек вершин, записанных в файле, имя которого вводится в процессе работы программы.

```
; Программа  построения графика функции f(x, y), заданной
; с равномерным шагом в виде квадратной матрицы  чисел,
; расположенных по столбцам и записанных в отдельном файле
; Для использования в 12-й версии Автокада добавлен режим ввода
; данных по запросам программы через диалоговые окна
;*********************************************************
(defun slider_action (val why)
          (if (or (= why 2) (= why 1))
          (set_tile "n" val))
)
(defun ebox_action (val why)
(if (or (= why 2) (= why 1))
          (set_tile "n_s" val))
)
```

Первые две функции предназначены для формирований значений переменных, используемых в скользящей шкале и информационном окне. Затем идет тело программы, в которой инициализируется диалоговое окно. Раздел, связанный с диалоговым окном, выделен в рамку.

```
(defun C:plot3d ()
```

Первой функцией вызывается функция вызова стандартного для Автокада диалогового окна выбора файлов:

```
(setq f (open (getfiled "Выбор исходного файла" "" "res" 2) "r"))
```

Затем загружается прилагаемая в стандартной поставке функция fplot:

```
(if (= fplot nil) (load "fplot"))
```

```
; Загрузка диалогового окна
(setq dcl_id (load_dialog "plot3d"))
```

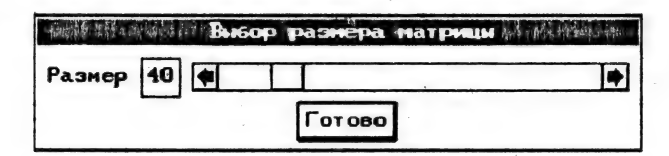
```
(if (not (new_dialog "plot3d" dcl_id)) (exit))
; Активизация переменных
 (action_tile "n_s" "(slider_action $value $reason)")
(action_tile "n" "(ebox_action $value $reason)")
(action_tile "accept"
        (strcat "(progn (setq msz (atoi (get_tile \"n\")))"
        "(done_dialog))"))
; Запуск диалога
(start_dialog)
; Выгрузка диалога
(unload_dialog dcl_id)
(initget 7)
(defun z(x y)
 (atof (read-line f))
)
(fplot 'z '(-2 2) '(-2 2) msz)
(close f)
)
;************************************************************
```

Текст описания диалогового окна (DCL-файл) приведен ниже:

```
/* plot3d.DCL - файл управления диалогом для plot3d.LSP */
plot3d : dialog {
/*      label = "Выбор размера матрицы";
        : row {
                : edit_box {
                label = "Размер";
                value = 40;
                edit_width = 2;
                fixed_width =true;
                key = "n";}
                : slider {
                        max_value = 99;
                        min_value = 25;
                        small_increment = 1;
                        big_increment = 5;
                        key = "n_s";
                        width = 46;
                        value = 40;}
                        }
        : button {
        label = "Готово";
        key = "accept";
        fixed_width =true;
        alignment = centered;
        is_default =true;}}
```

Описание каждой части диалогового окна “прозрачно”.

Выведенное на экран окно выглядит следующим образом:



Изменение размера матрицы может производиться либо перемещением квадратика скользящей шкалы с немедленным изменением значения числа в информационном прямоугольничке, либо активизацией информационного прямоугольничка и вводом туда конкретного числа, что приведет к изменению положения квадратика скользящей шкалы.

Меню Автокада и его настройка

В меню Автокада вызываются как базовые, так и разработанные пользователем команды и макроопределения (вставленные в меню последовательности команд, соответствующие данной задаче, или программные коды на языках Автокад и DIESEL. Из макросов можно вызывать пакетные файлы или Лисп- и СРП-программы). По сути, все разработанные для Автокада приложения представляют собой набор Лисп- или СРП-программ и разработанных под них меню, с помощью которых они вызываются при работе. Файлы меню представляют собой обычные текстовые файлы, содержащие командные строки и макроопределения Автокада. Исходные файлы имеют расширение .mnu, компилированные файлы имеют расширение .mnx. При первоначальной загрузке исходного файла он компилируется в файл .mnx. После редактирования исходного файла и последующей его загрузки он компилируется заново. Это происходит на основе отслеживания времени создания файла .mnu и файла .mnx. Если этого не происходит на вашем компьютере автоматически, обратите внимание на системные часы компьютера.

В 12-й версии Автокада описание макроопределений для меню может помещаться не в файл меню, как это делалось в более ранних версиях, а в отдельный файл с тем же именем и расширением .mnl, который загружается в память при загрузке файла меню.

4.1. Структура меню

Файлы меню делятся на разделы, относящиеся к определенным зонам меню (экранное и падающие меню). Разделы меню могут содержать субменю, которые листаются по экрану с помощью ссылок. Командные строки и макроопределения, вызов которых приводит к выполнению команды, называются пунктами меню. Пункты меню имеют заголовки. Хотя по структуре и действию все пункты меню аналогичны, в каждом разделе меню используется свой собственный синтаксис меток заголовков.

Зоны меню

Меню Автокада подразделяется на зоны:

- экранного меню;
- падающих и курсорного меню;

- графических меню;
- кнопочных меню устройства указания;
- планшетных меню;
- дополнительного меню устройства (системной мыши).

Файл меню может содержать разделы для каждой зоны меню. Соответствующие разделы загружаются в различные зоны. Например, пункты экранного меню загружаются в зоне экранного меню. Если планшетные меню настроены, загруженные новые планшетные меню становятся доступными из соответствующих зон планшета. Если на устройстве указания имеется несколько кнопок, кнопки, не используемые для выбора объектов, можно использовать для выбора пунктов непосредственно из нового загруженного кнопочного меню.

В файле меню может и не содержаться входа для каждого раздела меню. Это позволяет экспериментировать с каждым разделом независимо от других и добавлять только разделы, необходимые для конкретного приложения.

Метки раздела

Разделы файла меню начинаются с меток. Каждому разделу соответствует свой заголовок, после которого следует последовательность команд раздела меню.

Метка раздела	*Зона меню*
***BUTTONSn	Кнопочные меню устройства указания (n - число от 1 до 4)
***AUXn	Дополнительное кнопочное меню (n - число от 1 до 4)
***POPn	Падающие и курсорное меню (n - число от 0 до 16)
***ICON	Графические меню
***SCREEN	Экранное меню
***TABLETn	Планшетные меню (n - число от 1 до 4)

Для Автокада 10-й и 11-й версий число кнопочных и дополнительных кнопочных меню ограничено 1, число падающих меню - 10, и курсорное меню (с индексом 0) отсутствует. Метки указывают на начало раздела.

Пример:

```
***SCREEN
[Help]Help
[Quit]Quit Yes
***TABLET1
Line
```

circle
***BUTTONS
erase
oops

Для русифицированного Автокада команды и их опции (не заключенные в квадратные скобки надписи) должны записываться на русском языке. Для 12-й версии Автокада они уже могут быть английскими, но с предварительным подчеркиванием. Если в файле нет метки ***SCREEN, Автокад действует так, как если бы эта метка предшествовала первому пункту меню Автокада. При этом такой конкурирующий раздел будет работать также, как и экранное меню.

Субменю

Раздел меню может быть очень большим и не помещаться в выделенные зоны экрана. В этом случае он разделяется на субменю - более мелкие группы пунктов меню, расположенные внутри раздела меню. Например, выбор пункта EDIT экранного меню приводит к появлению на экранном меню команд редактирования. В зоне экранного меню пункты субменю заменяют все текущее меню или его часть. Субменю могут образовывать вложенные структуры. Механизм листания меню для различных разделов разный.

Метка субменю определяет начало подраздела в файле меню. Она имеет следующий формат:

**имя-меню [номер]

Метка имя-меню может содержать до 31 символа и включать буквы, цифры и специальные символы "$"(доллар), "_"(подчеркивание) и "-"(дефис). В метке не должно быть пробелов. Необязательное целое число номер задает номер начальной строки субменю. Пункты субменю непосредственно за меткой относятся к этому субменю. Разделы меню и субменю можно вызывать из других разделов.

Обращение к субменю выполняется следующим образом:

$раздел=субменю

где $ - специальный символ, используемый для указания загрузки раздела меню. В 12-й версии запись $M= позволяет пункту меню вызывать строковый DIESEL-макрос; раздел - указывает раздел меню. Допустимы следующие имена:

Имя	*Меню, для которого используется данное имя*
S	SCREEN
P1-P16	POP от 0 до 16
I	ICON
B1-B4	BUTTON от 1 до 4

T1-T4	TABLET от 1 до 4
A1-A4	AUX от 1 до 4

субменю - указывает имя активизируемого субменю по метке этого субменю.

Примеры:

```
$S=DRAW1
$T1=EDITCMDS
$I=HATCH1
```

Перед активизацией субменю копируется в стек. Например, если пункт меню содержит команду $S=PARTS, то активные пункты на экране копируются в экранный стек, а пункты субменю **PARTS активизируются. Чтобы восстановить предыдущие пункты экрана пункт меню должен содержать команду

```
$S=
```

без какой-либо метки субменю. При этом происходит извлечение из стека загруженных в последний раз пунктов, т. е. их повторная активизация. Допускается до 8 вложенных запросов субменю. Активизацию субменю можно осуществлять в ходе выполнения команды без ее прерывания. Например, последовательности команд

```
$S=ARC=ДУГА
ДУГА $S=ARC
```

За именем субменю должен следовать пробел для отделения его от последующих команд в пункте меню.

Пример файла меню:

```
***SCREEN
[ПРИМЕР]
                          ; пустая строка
[РИСУЙ...]$S=Draw_Root
[Редакт...]$S=Edit_Root
                          ; пустая строка
[Пока]конец
                          ; три пустые строки заполняют страницу
                          ; меню до 10 строк и служат для
                          ; замещения (перекрытия) пунктов субменю.
[-Главн-]$S=SCREEN
                          ; Так как ни одно из субменю не опускается ниже этой
                          ; строки, она отображается во всех меню и служит
                          ; для вызова основного меню.
**Draw_Root 2
                          ; 2 после имени субменю начинает его с пустой
                          ; строки после строки [ПРИМЕР]
[Отрезок:]Отрезок
[Круг:    ]Круг
```

```
[Дуга:      ]Дуга
                                    ; Не менее одной пустой строки
**Edit_Root 2


[Сотри:     ]$S=Sel_Obj Сотри
[Копируй:  ]$S=Sel_Obj Копируй
[Перенеси]$S=Sel_Obj Перенеси


**Obj_sel 2
**Sel_Obj 2
                                    ; Для вызова этого меню можно использовать имена
                                    ; и Obj_sel и Sel_Obj.
[Последний]Последний
[Текущий]Текущий
[Рамка]Рамка
[Секрамка]Секрамка

[- ПРЕД-]$S=
$S= вызывает предыдущее меню
***BUTTONS1                         ; присваивает "Enter" кнопке 2
Освежи                              ; присваивает команду Освежи" кнопке 3
```

В данном примере приведены три субменю: 'Draw_Root, Edit_Root и Sel_obj. Субменю Draw_Root и Edit_Root вызываются из основного экранного меню при выборе пунктов Рисуй или Редактируй. Субменю Draw_Root позволяет выбрать три пункта, соответствующие командам Автокада. Субменю Edit_Root также содержит три пункта, каждый из которых включает субменю Sel_obj перед выполнением соответствующей команды.

Во всех меню в первой строке появляется пункт -ГЛАВН-, выбор которого позволяет возвратиться в главное экранное меню. Каждое экранное меню перекрывает лишь столько строк, сколько строк в нем содержится. Если экранное меню содержит больше пунктов, чем строк на экране, или кнопочное меню содержит больше пунктов, чем имеется кнопок, лишние строки игнорируются. Для удлинения субменю с целью перекрыть предыдущие меню можно использовать пустые строки.

Синтаксис пунктов меню

Каждый пункт меню может состоять из необязательного заголовка, команды или последовательности команд и параметров; обычно каждый пункт меню находится на одной строке. Если командные параметры (опции) присутствуют в строке меню, будьте внимательны и проверьте последовательность их задания вводом команды из командной строки. Все символы строки являются значимыми, даже пробелы. Следите за

тем, чтобы после заголовка пункта меню не было пробела! Заголовки - это набор символов, который заключается в квадратные скобки. Заголовок не обязательно повторяет имя команды. В каждом типе меню заголовки обрабатываются по-разному. Текст заголовков для кнопочных, планшетных и дополнительных меню не отображается и используется в качестве комментария в файле меню.

Введенная в 12-й версии системная переменная MENUCTL управляет автоматической сменой экранных меню при вызове соответствующих команд. Если она установлена в единицу (Вкл) и команда вызывается из пункта меню, происходит выполнение выражения S=имя_команды, которое вызывает экранное субменю с именем, соответствующим имени команды (если такое существует). В стандартном меню acad.mnu переменная ENUCTL устанавливается в единицу из файла acad.mnl.

При выборе пункта меню Автокад автоматически ставит после него пробел. Например, если имеется пункт меню

LINE

то Автокад воспринимает его как команду LINE Spacebar. Иногда это нежелательно. Например, в конце текстовой строки, создаваемой командами TEXT или IMENSION, должен стоять не пробел, а ENTER. В других случаях для завершения команды необходимо использовать более одного пробела (или ENTER), однако некоторые текстовые редакторы не позволяют создавать строку, завершающуюся пробелами. Для решения этих проблем приняты следующие соглашения:

- если в пункте меню появляется точка с запятой (;), Автокад заменяет ее на ENTER;
- если строка заканчивается каким-либо управляющим символом, обратной косой чертой (\), знаком плюс (+) или точкой с запятой (;), Автокад не добавляет после него пробела.

Если в процессе выполнения пункта меню требуется ввести информацию с клавиатуры или устройства указания, вводится обратная косая черта. После нее пробел не ставится.

Пример:

[Круг]Круг \1

Здесь запрашивается положение центра, затем считывается единица как радиус.

Обычно после ввода одного параметра пункт меню продолжает выполняться. Поэтому конструкция пункта меню, в которой допускается ввод произвольного числа параметров (как при выборе объектов) и затем продолжение каких-либо операций, невозможна. Это ограничение не распространяется на команду ВЫБЕРИ (SELECT): в этом случае обратная косая черта задерживает выполнение следующей операции до

тех пор, пока набор не будет полностью сформирован, т. е. пока не будет нажата клавиша "ENTER".

Пример:

[Yellow]Select \change previous ; properties color yellow ;

Здесь используется команда ВЫБЕРИ (SELECT) для формирования набора из одного или нескольких объектов. Затем выполняется команда ИЗМЕНИ (CHANGE), производится обращение к набору "Текущий" ("Previous") и цвет выбранных объектов изменяется на желтый.

Так как обратная косая черта приостанавливает выполнение пункта меню, ее нельзя использовать для других целей; при необходимости ввода пути доступа к файлам в меню следует использовать символ прямой косой черты: /support/acad.mnu.

Выбранный пункт меню *не возобновляется*:

- до завершения выбора в режиме объектной привязки, если ожидается ввод точки;
- до завершения ввода всех координат точки, если используются координатные фильтры;
- до завершения всего набора в команде ВЫБЕРИ (SELECT);
- при вводе "прозрачной" команды - до ее выполнения и ввода Enter;
- если в качестве ответа указывается другой пункт меню (для выбора опции или выполнения "прозрачной" команды) до выполнения этого пункта (в свою очередь, новый выбранный пункт может иметь свои символы обратной косой черты).

12-я версия Автокада отличается от предыдущих еще и тем, что при вводе команд и их опций на английском языке они автоматически переводятся на русский, если перед командой или ключевым словом стоит символ подчеркивания (_). Это справедливо и для меню, и для ввода команд с клавиатуры.

Пример фрагмента стандартного меню Автокада русскоязычной версии:

```
***POP0
[Привязка]
[БЛИжайшая]_nea
[КАСательная]_tan
[КВАдрант]_qua
[КОНточка]_endp
[НОРмаль]_per
[ПЕРесечение]_int
[СЕРедина]_mid
[ТВСтавки]_ins
[УЗЕл]_nod
[ЦЕНтр]_center
[НИЧего]_non
```

Для отмены действия команды из командной строки мы используем ввод Ctrl+C. В этом случае команда отменяется на один уровень: если Автокад выполняет основную команду, она отменяется; если выполняется "прозрачная" команда внутри основной, ввод Ctrl+C отменяет "прозрачную" команду; если выполняется сложная команда, например DIMENSION (РАЗМЕР) и в ней команда, например, ANGLE (УГЛОВОЙ), а в ней еще и "прозрачная" команда, например ZOOM (ПОКАЖИ), то для отмены всей этой гирлянды требуется ввод трех последовательностей Ctrl+C. В меню такая последовательность действий выполняется вводом нужного числа символов ^C без пробелов.

Кроме управляющего символа ^C в Автокаде могут использоваться и другие управляющие символы, которые выполняют определенные действия, например включение и выключение сетки, режима орто, перехода на другую плоскость изометрии и т. д. Неалфавитные управляющие коды ASCII отмечаются символом (^), за которым следует еще один символ:

^@	(ASCII код 0)
^[	(ASCII код 27)
^\	(ASCII код 28)
^]	(ASCII код 29)
^^	(ASCII код 30)
^_	(ASCII код 31)

Использование комбинации ^H, которая применяется для стирания символа слева от курсора, дает возможность формирования "цифровой клавиатуры":

[1]1x^H
[2]2x^H
[3]3x^H
[4]4x^H
...
[Дес. точка].^H
[Ввод]

При выборе соответствующей строки меню вводится требуемая цифра или десятичная точка, а, поскольку все строки, кроме последней, заканчиваются управляющим символом, к ним не добавляется ни пробела, ни "ENTER". Последовательно введя

[2],[3],[Дес. точка],[1],[4],[Ввод]

получим число 23.14.

Если пункт меню не помещается в одну строку, его можно продолжить на следующей строке, поставив в конце предыдущей знак плюс (+). Число строк продолжения не ограничивается.

Выполнение каждой команды заканчивается приглашением к выполнению следующей команды. Иными словами, если выполнена одна команда, Автокад приглашает ввести следующую высвечиванием

сочетания Command: (Команда:) в командной строке. При необходимости повторного вызова только что выполнившейся команды, вызванной из какого-либо раздела меню, такая строка меню должна начинаться звездочкой (*). Именно этот механизм используется в падающем меню DRAW (РИСУЙ). Функция повтора не действует в пунктах графических меню.

В командах редактирования опции выбора объектов можно ограничить выбором только одного объекта, указав после вызова команды ключевое слово "Единственный" или "_Single" (в английской версии - "Single"). В качестве примера можно привести пример формирования команды ERASE (СОТРИ) при выборе одного объекта и его немедленного стирания без дополнительных действий:

```
[Сотри один]*^C^C_Erase _Single
[Сотри один]*^C^CСотри Единственный
```

Обе строки в русскоязычной версии Автокада работают одинаково. Звездочка перед командой приводит к повторному вводу команды СОТРИ, работающей в выбранном режиме.

Перечень управляющих последовательностей символов, используемых в файлах меню:

Символы	*Описание*
***	Указывает начало заголовка
**	Указывает метку раздела меню
[ ]	Выделение заголовков для пунктов и разделов меню
; или ^M	Ввод клавиши "Enter" или "Spacebar"
^I	Ввод клавиши "TAB"
<пробел>	Пробел между последовательностями команд и их опций эквивалентен вводу Enter
\	Приглашение к вводу информации пользователем
_(подчеркивание)	Перевод на русский язык английской команды или опции
+	Продолжение набора команд на следующей строке
=*	Вывод текущей пиктограммы, падающего или курсорного меню на экран
*^C^C	Повтор команды, вызванной из меню, после выполнения
$	Загрузка раздела меню или ввод условного выражения языка DIESEL ($M=)

^B	Переключение режима Шаг
^C	Отмена команды
^D	Переключение режима отображения координат
^E	Переход на следующую плоскость изометрии
^G	Переключение режима Сетка
^H	Удаление предыдущего символа
^O	Переключение режима Орто
^P	Переключение системной переменной MENUECHO
^Q	Переключение эхо-вывода на принтер
^T	Переключение режима Планшет
^V	Изменение текущего видового экрана

Кнопочные и дополнительные меню

Кнопочное (BUTTONS и BUTTONS1-4 для 12-й версии Автокада) и дополнительные меню (AUX и AUX1-4 для 12-й версии Автокада) аналогичны по формату. Их использование зависит от типа используемого системой устройства указания. Если у компьютера нет системной мыши (встроенного устройства указания), имеющееся устройство будет использовать кнопочные меню. Системная мышь использует дополнительные меню, а другое устройство будет использовать кнопочные меню.

По описанию кнопочные и дополнительные меню аналогичны. Доступ к каждому кнопочному меню можно осуществлять путем нажатия последовательности клавиш и кнопок:

Последовательности кнопок и клавиш	*Кнопочные меню*
Нажатие одной кнопки	BUTTONS1
Нажатие клавиши Shift+кнопка	BUTTONS2
Нажатие клавиши Ctrl+кнопка	BUTTONS3
Нажатие клавиши Shift+Ctrl+кнопка	BUTTONS4

Для совместимости с 10-й и 11-й версиями меню BUTTONS аналогично меню BUTTONS1. Если в меню имеются разделы BUTTONS и BUTTONS1, в 12-й версии предпочтение отдается BUTTONS1.

Раздел BUTTONS1 кнопочного меню в стандартном файле acad.mnu выглядит следующим образом:

```
***BUTTONS1
;
```

```
$p0=*
^C^C
^B
^O
^G
^D
^E
^T
```

В этом разделе кнопка указания не программируется, а первая строка, т. е. точка с запятой, относится к программированию второй кнопки. Кнопка выбора не перепрограммируется во всех меню! Устройство указания будет распознавать только те строки, для которых имеются кнопки. Способ кодирования команд для кнопок аналогичен только что рассмотренным. В кнопочных меню заголовки в квадратных скобках рассматриваются как комментарии. При выборе кнопочного меню Автокад воспринимает не только номер кнопки, но и координаты перекрестия на экране. Эти координаты можно использовать в выбранной команде. Для этого используется символ обратной косой черты, которая в этом случае позволяет ввести сразу после команды зафиксированную точку:

```
***BUTTONS
LINE
LINE \
```

Второй кнопке запрограммирован ввод команды ОТРЕЗОК (LINE) с игнорированием точки указания, при нажатии третьей кнопки изображение отрезка начнется из точки указания и команда продолжит диалог приглашением К точке: (To point:).

Экранные меню

Раздел экранного меню управляет зоной бокового экранного меню на экране Автокада. Если в файле меню нет этого раздела, Автокад присваивает этой зоне текст меню, расположенный в начале файла. Экранные меню не ограничены по размеру.

Метка раздела ***SCREEN означает начало экранного меню Автокада. Метка **S указывает раздел субменю. В начале сеанса редактирования на дисплее Автокада в зоне экранного меню появляется корневое меню, т. е. страница с меткой S.

Раздел экранного меню ... ... и его отображение на экране:

***SCREEN

```
**S
[Автокад]^C^C^P(ai_rootmenus) ^P
[* * * *]$S=OSNAPB
[БЛОКИ]$S=X $S=BL
[ДИСПЛЕЙ]$S=X $S=DS
[ЛИСП]$S=X $S=LP
[НАСТРОЙ]$S=X $S=SET
[ПАРАМЕТ]$S=X $S=ACADENCE
[ПОВЕРХН.]$S=X $S=3D
[ПСК:]^C^C_UCS
[РАЗМЕР:]^C^C_DIM
[РАС]^C^C^P(ai_aseinit_chk) ^P
[РЕДАКТ]$S=X $S=ED
[РИСУЙ]$S=X $S=DR
[СВИД]$S=MVIEW
[СЛОЙ...]$S=LAYER '_DDLMODES
[СПРАВКИ]$S=X $S=INQ
[ТЕЛА]$S=X $S=SOLIDS
[ТОНИР]$S=X $S=RENDER
[УТИЛИТЫ]$S=X $S=UT
[ЧЕРТИ...]^C^C_PLOT

[СОХРАНИ:]^C^C_QSAVE
```

```
Автокад
* * * *
БЛОКИ
ДИСПЛЕЙ
НАСТРОЙ
ПОВЕРХН.
ПСК:
РАЗМЕР:
РАС
РЕДАКТ
РИСУЙ
СВИД
СЛОЙ. . .
СПРАВКИ
ТЕЛА
ТОНИР УТИЛИТЫ
ЧЕРТИ. . .
СОХРАНИ:
```

В Автокаде принято соглашение об окончании заголовков строк, соответствующих настоящим командам Автокада, двоеточием. Если посмотреть на представленные картинки, то мы увидим, что на первой странице меню всего несколько истинных команд: остальные строки управляют перелистыванием меню. Большинство строк первой страницы меню начинается конструкцией

```
$S=X
```

Это вызов страницы меню, которая просто готовит место под вывод нового набора команд. На каждой странице может присутствовать различное число строк. Поэтому, если страница короткая, на ней будут присутствовать остатки предыдущей страницы. Для предотвращения такого нежелательного наложения заготовлена специальная "очищающая" страница, которая имеет вид:

```
**X 3

; 15 пустых строк


[ПРЕДМЕНЮ]$S= $S=
[ РИСУЙ  ]^C^C$S=X $S=DR
[ РЕДАКТ ]^C^C$S=X $S=ED
```

Она начинается с третьей строки и заканчивается вызовами предыдущей страницы, страниц редактирования и рисования. Первые две строки любой страницы меню предназначены для вызова корневого меню и меню объектных привязок. Эти соглашения не рекомендуется изменять и при создании своих новых страниц меню.

Падающие и курсорное меню

Падающие меню располагаются в верхней части экрана. Они доступны только при наличии устройства указания, т. е., если к вашему компьютеру не подключена мышь или планшет, эти меню недоступны. В 10-й и 11-й версиях Автокада максимальное число падающих меню составляет 10, в 12-й версии их число увеличено до 16. Кроме того, в 12-й версии во многом изменился в сторону расширения удобств пользования интерфейс падающих меню. Это несколько усложнило способы их формирования и программированного обращения. Введение языка DIESEL теперь позволяет программно перестраивать заголовки падающих меню. Курсорное меню представляет собой модификацию падающего меню, которое вызывается одной из кнопок устройства указания (для двухкнопочной мыши это комбинация клавиш Shift + правая кнопка; для трехкнопочной мыши это вторая кнопка). В отличие от падающих меню, которые разворачиваются из строки меню, курсорное меню всегда появляется в точке перекрестия курсора на графическом экране. Заголовок курсорного меню на экране не появляется и, по сути, представляет собой просто комментарий в файле меню. Доступ к курсорному меню из других меню программ на Лиспе или Си производится с помощью команды SP0=*. При активном курсорном меню строка меню отключается.

В 12-й версии падающие меню могут содержать до 999 пунктов меню, а курсорное меню - до 499 пунктов. Оба предела включают все субменю в иерархии. Если падающие или курсорное меню не помещаются на графическом экране по длине, лишние пункты усекаются.

Каждый раздел падающего меню начинается с заголовка ***POPn, где n может принимать значения от 0 (для курсорного меню) до 16 в 12-й версии Автокада. Приведем символы, которые выполняют специальные функции в заголовках падающих или курсорного меню:

Символ	*Описание*
--	Заголовок пункта раздвигается, становясь разделительной линией (если используется без других символов)
+	Продолжает макроопределение на следующей строке (если в строке является последним)
->	Указывает на наличие субменю (12-я версия)
<-	Указывает на последний пункт в субменю (12-я версия)

<-<-...	Указывает на последний пункт субменю и завершает родительское меню (12-я версия)
S(	Вычисляет в заголовке строковый макрос языка DIESEL (если последовательность символов S (стоит в начале строки) (12-я версия)
!c	Помечает пункт меню специальным (не буква или цифра) символом. Последовательность !. (восклицательный знак с точкой) ставит специальный маркер (галочку). Поддерживается не на всех платформах (12-я версия)
/c	Указывает в заголовке клавишу акселератора меню. Поддерживается не на всех платформах (12-я версия)
<c	Указывет тип выделения шрифта. Допустимые типы: <B = полужирный, <O = контурный, <S = с оттенением, <I = курсивный. Поддерживается не на всех платформах (12-я версия)
^имя^	В заголовке отображается пиктограмма имя. Поддерживается не на всех платформах (12-я версия)

Обычно падающее меню появляется непосредственно под его заголовком. Но для отображения длинных пунктов крайние правые падающие меню сдвигаются влево. Максимальное количество пунктов в падающих меню зависит от драйвера монитора (например, 21 для обычных мониторов). Это надо учитывать при разработке прикладных меню.

```
[Средства]
[ПОМОЩЬ!]'?
[Отмена]^C^C^C
[--]
[Отмени]_U
[Верни]^C^C_redo
[--]
[Объектные фильтры...]'ФИЛЬТР
[->Объектная привязка]
 [--]
[->Справки]
 [Список]^C^C_list
 [Статус]'_status
 [--]
 [Площадь]^C^C_area
 [Дист]'_dist
 [<-Коорд]'_id
[--]
[Калькулятор]'кальк
```



Здесь показан фрагмент файла acad.mnu и вид части развернутого падающего меню на экране компьютера. В первой строке после разделителя ***POP2 расположен заголовок, который определяет данный раздел в свернутом виде. Длина заголовка не должна превышать 14 символов. Конечно, если вы хотите поместить все заголовки меню без искажений, заголовки следует делать короче. В названия не рекомендуется вставлять пробелы, поскольку пользователь не сможет различить заголовки разных меню. Если первая строка пустая, в строке меню заголовки не появляются и не вызываются. Остальные заключенные в квадратные скобки строки представляют собой заголовки пунктов меню. Эти строки могут быть любой длины, но не следует увлекаться длинными строками.

Строка меню, падающие или курсорное меню недоступны во время исполнения следующих команд:

- ДТЕКСТ (DTEXT) после установки угла поворота;
- ЭСКИЗ (SKETCH) (после установки значения приращения);
- ТЗРЕНИЯ (VPOINT) (при выдаче компаса и точки зрения);
- ПОКАЖИ Динамика (ZOOM D);
- ДВИД (DVIEW).

Графические меню

Раздел меню ***ICON представляет собой меню в виде графических окон. Число окон и их размеры, выводимые на экран, для различных версий Автокада различны: если для версий младше 12 минимальное число слайдов или команд в одном графическом экране равно 4, а максимальное - 16, то в 12-й версии число экранчиков графического меню фиксированно и равно 20. Зона графического меню разделена на графическую и текстовую части, взаимосвязанные между собой: при указании слайда подсвечивается соответствующая ему строка, и наоборот. Команды исполнения расположены в нижней части меню, и теперь

не требуется выделения места для их ввода в графической зоне. Число строк в каждом разделе графического меню не ограничено: Автокад сам разбивает их постранично по 20 на странице и предоставляет возможность перелистывания отдельных страниц автоматически или опциями "Следующий", "Предыдущий".

Фрагмент графического меню на экране выглядит следующим образом:

```
**vporti
[Настройка видовых экранов]
[acad(vport-1,Один)]^C^C(ai_tiledvp 1 nil)
[acad(vport-3v,Три: вертикал)]^C^C(ai_tiledvp 3 "V")
[acad(vport-3h,Три: горизонт)vp3h]^C^C(ai_tiledvp 3 "G")
[acad(vport-4,Четыре: равные)]^C^C(ai_tiledvp 4 nil)
[acad(vport-2v,Два:  вертикал)]^C^C(ai_tiledvp 2 "V")
[acad(vport-3r,Три: справа)]^C^C(ai_tiledvp 3 "R")
[acad(vport-3l,Три: слева)]^C^C(ai_tiledvp 3 "L")
[acad(vport-4l,Четыре: слева)]^C^C(ai_tiledvp 4 "L")
[acad(vport-2h,Два: горизонт)]^C^C(ai_tiledvp 2 "H")
[acad(vport-3a,Три: выше)]^C^C(ai_tiledvp 3 "A")
[acad(vport-3b,Три: ниже)]^C^C(ai_tiledvp 3 "B")
[acad(vport-4r,Четыре: справа)]^C^C(ai_tiledvp 4 "R")
```

В начале раздела находится заголовок, который изображается в самом верху графического меню. Затем располагаются строки команд со своими заголовками. В зависимости от конструкции заголовка поле списка заполняется по-разному:

Поле списка	*Результат вывода*
[имя_слайда]	В поле списка выводится имя слайда, который отображается в виде пиктограммы
[имя_слайда, заголовок]	В поле списка выводится текст заголовок, а слайд имя_слайда отображается в виде пиктограммы
[библиотека (имя_слайда)]	В поле списка выводится имя слайда из библиотеки слайдов библиотека, который отображается в виде пиктограммы
[Библиотека (имя_слайда, заголовок)]	В поле списка выводится текст заголовок, слайд имя_слайда из библиотеки, библиотека отображается в виде пиктограммы
[пустой]	При задании текста пустой заголовок используется как разделительная строка в поле списка и никакая пиктограмма не отображается

[заголовок]	Если первым символом в заголовке пункта является пробел, в поле списка выводится текст заголовок и никакая пиктограмма не отображается. Это позволяет использовать команды и пункты типа Выход без необходимости создания слайдов с соответствующими словами

В 10-й и 11-й версиях Автокада поле списка отсутствует. Поэтому создание графических меню в этих версиях более простое и с меньшими возможностями. Приходится ограничивать раздел меню 16 строками, выводя команды типа "Предыдущий", "Следующий", "Выход" через [заголовок] с первым пробелом и обеспечивать листание "страниц" графического меню через вызовы следующих разделов программно - явным образом.

Настройка видовых экранов

Один
Три: вертикал
Три: горизонт
Четыре: равные
Два: вертикал
Три: справа
Три: слева
Четыре: слева
Два: горизонт
Три: выше
Три: ниже
Четыре: справа

Предыдущ. | Следующ. | Да | Отмена

Графическое меню вызывается парой команд

$I=имя_раздела
$I=*

Эти команды могут присутствовать в любых разделах меню, но их нельзя ввести с клавиатуры. Таким образом, первичный вызов графического меню нужно обеспечивать либо из экранного, либо из падающего меню. В дальнейшем вызовы последующих графических меню можно осуществлять и из предыдущих графических меню. На сложность создаваемых таким образом структур принципиальных ограничений нет, главное здесь - не запутать пользователя.

4.2. Подготовка слайдов для графических меню

Любой слайд, созданный в Автокаде с помощью команды ДСЛАЙД (MSLIDE), можно использовать в качестве пиктограммы для меню. Для оптимального использования графических меню следует придерживаться следующих правил:

- слайд должен быть достаточно простым, чтобы уменьшить время генерации пиктограмм на экране. Лучше использовать упрощенные пиктограммы и не стараться получить слайд с подробной проработкой деталей;
- изображение пиктограммы в графическом меню занимает относительно небольшую площадь. Старайтесь использовать ее рационально, заполняя всю зону картинкой. При подготовке слайда рисунок должен находиться в центре экрана и по возможности заполнять его полностью. Соотношение размеров пиктограмм в меню соответствует соотношению ширина : высота = 1.5 : 1. При работе на видовом экране с таким соотношением его размеров можно разместить изображение так же, как оно будет выглядеть на пиктограмме;
- закрашивание областей на экране занимает много времени. Лучше не использовать закрашивание и отключать режим закрашивания при использовании широких полилиний и фигур;
- пиктограммы должны быть понятными. Не пытайтесь закодировать абстрактные представления. Пользователь средних способностей не должен ломать голову над возможными последствиями применения команды.

4.3. Планшетные меню

Если у вас есть планшет, его можно использовать для ввода команд Автокада. Если рассмотреть вопрос, с чем удобнее работать - с мышью или планшетом, то тут все сводится к тому, к чему вы привыкли. На планшете стандартной конфигурации собраны не все команды Автокада, а только наиболее часто используемые. Конечно, при хорошем навыке работать с планшетом можно быстрее, чем с мышью, поскольку доступ к командам осуществляется без листания “страниц” экранного или падающих меню. Так что если вам приходится заниматься большими объемами графических построений, то планшет имеет смысл приобрести и использовать не только как инструмент для ввода чертежей в режиме скалывания, но и как устройство ввода команд.

Автокад позволяет настроить до четырех областей планшета в качестве зон меню для ввода команд. Разделы файла меню с метками TABLET1-4 определяют макросы меню, связанные с определенным положением зон указания относительно координат планшета.

В пунктах планшетного меню используется синтаксис, аналогичный используемому в других разделах. Заголовки пунктов рассматриваются как комментарии. Автокад способен распознавать до 32766 пунктов меню в каждом разделе планшета.

В стандартном меню acad.mnu в разделе TABLET1 заголовки помечают клетки темплета, прилагаемого к пакету. Команды в этих клетках отсутствуют, да и сам темплет в этом разделе "прозрачен". Вы можете разместить свои собственные команды и макроопределения в эту зону и изобразить разбитый на клетки чертеж нужного размера для ввода своих команд и макроопределений. Строки ниже [I-25] модифицировать не следует.

4.4. Использование Автолиспа в макроопределениях

Использование Автолиспа при создании макроопределений в меню позволяет:

- сохранять данные в переменных, обрабатывать данные и передавать их Автокаду;
- управлять отрисовкой примитивов посредством установки системных переменных Автокада, сохранения системных переменных, запрашиванием настроек и изменением их в соответствии с ответами пользователя;
- устанавливать переменные в процессе выполнения команд (в "прозрачном" режиме) для более полного удовлетворения запросов пользователя;
- устанавливать ответы по умолчанию на запросы Автокада и использовать их в командах и макроопределениях;
- считывать и изменять геометрические данные объектов, используя данные в своих программах;
- получить доступ к справочной информации, содержащейся в таблицах Автокада, например об установке слоев, блоков, стилях текста и типах линий.

В этом разделе будет показано на примере, как можно воспользоваться средствами Автолиспа для модификации меню Автокада. Вообще-то использование Автолиспа при изменении меню не обязательно - меню может и не обращаться к Автолиспу, используя только стандартные команды Автокада. Однако практика показывает, что на каком-то

этапе развития вашего меню пользоваться средствами Автолиспа вам все-таки придется. Кроме того, даже если ваш пункт меню совсем прост и использует только команды Автокада, все равно желательно при помощи Автолиспа сохранять установленные по умолчанию значения используемых системных переменных Автокада и восстанавливать их по окончании работы вашего подменю. Это позволит вам не беспокоиться об изменении установок Автокада или восстанавливать их вручную.

Для примера решим следующую задачу. Если вы устанавливали в своей системе редактор Norton Editor так, как было показано нами ранее, то, возможно, нашли утомительным каждый раз при вызове редактора задавать ему имя файла, которое надо указывать полностью - с расширением .lsp. Кроме того, при ошибке в имени файла этот редактор не выдает никакого сообщения, а просто открывает новый файл с тем именем, которое было ему указано. Поставим себе цель организовать подменю Автокада, названное, скажем, LISP, в котором можно было бы, один раз задав имя файла-программы на Автолиспе, многократно вызывать его на редактирование. Желательно, чтобы имя файла можно было задавать без расширения, так как оно в нашем случае меняться не будет.

Решение лежит на поверхности: нужно написать на Автолиспе программу (а если быть более точным, то это макроопределение, поскольку оно будет вставлено в меню), которая запоминала бы один раз указанное имя файла в некоторой текстовой переменной. Поскольку внешним командам как и любым другим командам Автокада, можно из Автолиспа передавать параметры, нет ничего проще, чем автоматизировать этот процесс.

Ниже показано, как можно это сделать. Придется внести изменения в меню Автокада (файл acad.mnu). Поместим подменю LISP в экранное (SCREEN) меню Автокада. Это делается добавлением в экранное меню всего одной строки:

```
***SCREEN
**S
[AutoCAD]^C^C$S=X $S=S $P1=POP1 $P3=POP3
[* * * *]$S=OSNAPB
[Setup]^C^C^P(progn(prompt "Loading setup...")(load "setup")) ^P$S=X+ $S=UNITS
[BLOCKS]$S=X $S=BL
[DIM:]$S=X $S=DIM ^C^CDIM
[DISPLAY]$S=X $S=DS
[DRAW]$S=X $S=DR
[EDIT]$S=X $S=ED
[INQUIRY]$S=X $S=INQ
[LAYER:]$S=X $S=LAYER ^C^CLAYER
[SETTINGS]$S=X $S=SET
[PLOT]$S=X $S=PLOT
```

```
[UCS:]$S=X $S=UCS1 ^C^CUCS
[UTILITY]$S=X $S=UT
[LISP]$S=X $S=LP
[3D]$S=X $S=3D
[ASHADE]^C^C^P(progn(setq m:err *error*)(prin1))(defun *error* (msg)+
(princ msg)(setq *error* m:err m:err nil)(princ))(cond ((null C:SCENE)(vmon)+
(if (/= nil (findfile "ashade.lsp"))(progn (terpri);+
(prompt "Please wait...  Loading ashade.  ")(load "ashade")+
(menucmd "S=X")(menucmd "S=ASHADE")(setq *error* m:err m:err nil))+
(progn (terpri);(prompt "The file 'Ashade.lsp' was not found in your current+ search+
directories.")(terpri)(prompt "Check your AutoShade Manual+
for installation instructions.");(setq *error* m:err m:err nil)(princ))))+
(T (setq *error* m:err m:err nil)(menucmd "S=X")+
(menucmd "S=ASHADE")(princ))) ^P
[SAVE:]^C^CSAVE
```

Как видим, новое подменю отличается от стандартного только одной строчкой:

```
[LISP]$S=X $S=LP
```

С ее помощью определяется подменю LP, в котором будут находиться добавляемые нами средства. Переход в это подменю осуществляется при выборе пункта LISP стандартного экранного меню Автокада. Теперь самое главное - написание собственно макроопределений меню:

```
**LP 3
[Lisp]^C^C^P (progn (terpri) (setq lispfile (strcat (setq lispfile (getstring +
"Input Lisp file name (without extension): ")) ".lsp")))
[File:]^C^C^P (progn (terpri) (setq lispfile (strcat (setq lispfile+
(getstring "Input Lisp file name (without extension): ")) ".lsp")))
[EDIT:]^C^C^P(progn (if (/= lispfile nil) (progn (terpri) (setq lispfile (strcat +
(setq lispfile (getstring "Input Lisp file name (without extension): "))+
 ".lsp"))) (command "LEDIT"))
[LOAD:]^C^C^P (if (= lispfile nil) (progn (terpri) (setq message +
"Nothing to load: Enter file name first!") (terpri) (setq a nil)) (progn (terpri)+
(load lispfile))^P
[LIST:]^C^C^P(command "DIR" "*.lsp") ^P
```

Всего пунктов четыре: Lisp File, EDIT и LOAD. Самый простой из них - пункт LOAD, осуществляющий загрузку в Автокад программы на Автолиспе. Загружается та программа, с которой идет работа в настоящий момент. Пункт Lisp File (фактически это один и тот же пункт, который активизируется выбором как Lisp, так и File) запоминает в строковой переменной lispfile вводимое пользователем имя (без расширения) текущего файла Автолиспа. После ввода имени файла к нему подсоединяется расширение .lsp. При выборе пункта меню EDIT автоматически выполняется внешняя команда EDIT, которой передается в качестве параметра имя файла на Автолиспе.

В пунктах EDIT и LOAD нужно предусмотреть проверку имени файла на тот случай, если пользователь выберет этот пункт до того, как будет задано имя файла. При этом имя файла запрашивается и в пункте EDIT, а в пункте LOAD выдается сообщение об ошибке.

Описанная выше процедура кажется сложной, однако, один раз ее проделав, вы сэкономите себе много времени при работе с Автолиспом. В этом и заключается смысл автоматизации пользовательских приложений.

Для более удобного и эффективного использования Автолиспа в меню в 12-й версии Автокада можно разместить Лисп-процедуры, используемые в меню, в отдельном файле с расширением .mnl - это делает меню более компактным. Имя этого файла должно совпадать с именем соответствующего меню. Автокад автоматически загружает файл .mnl при загрузке файла меню (.mnu или .mnx) с таким же именем.

Язык строковых выражений (DIESEL)

Язык строковых выражений (DIESEL - Direct Interpretively Evaluated String Expression Language) можно использовать для формирования текстовой строки, которая с помощью системной переменной MODEMACRO выводится в статусную строку, или изменять заголовки в пунктах меню в процессе его работы без редактирования и перезагрузки. Это достаточно медленный интерпретирующий язык, однако он надежен и хорошо выполняет свою работу. DIESEL-выражения работают со строками - на входе и выходе функций этого языка всегда будут строки.

По сути, с помощью команд языка DIESEL вы, в зависимости от хода процесса проектирования, можете видоизменять заголовки строк меню, выводимые на экран; выполняемые при этом команды следует изменять при помощи функций Автолиспа. Эти возможности имеются только в 12-й версии Автокада. В 10-й версии в статусную строку можно было записать информацию с помощью функции Автолиспа GRTEXT с указанием опции -1. Эта информация будет присутствовать в текстовой строке до тех пор, пока ее не перепишет сам Автокад, например при изменении координат точки курсора или имени текущего слоя.

5.1. Строковые функции языка DIESEL

Признаком ввода DIESEL-выражения служит символ S (денежная единица доллар). Все функции DIESEL начинаются с этого символа. Затем следует круглая скобка, в которой перечисляется список, первым элементом которого будет имя выполняемой функции. Длина списка ограничивается 10 элементами, т. е. каждая функция не может иметь более 9 параметров. Элементом списка может быть другая DIESEL-функция. Превышение числа параметров приведет к выдаче сообщения об ошибке. В Автокаде разрешены следующие функции DIESEL-языка:

$(+, значение1, значение2 [, значение3, ..., значение9])

Возвращается сумма чисел *значение1, значение2, ..., значение9.*

$(-, значение1, значение2 [, значение3, ..., значение9])

Возвращается результат вычитания чисел из числа *значение1* чисел *значение2, ..., значение9.*

$(, значение1, значение2 [, значение3, ..., значение9])*

Возвращается произведение чисел *значение1, значение2, значение3, ..., значение9.*

$(/, значение1, значение2 [, значение3, ..., значение9])

Возвращается результат деления числа *значение1* на числа *значение2, ..., значение9.*

$(=, значение1, значение2)

Если числа *значение1* и *значение2* равны, возвращается 1, в противном случае 0.

$(<, значение1, значение2)

Если число *значение1* меньше числа *значение2*, возвращается 1, в противном случае 0.

$(>, значение1, значение2)

Если число *значение1* больше числа *значение2*, возвращается 1, в противном случае 0.

$(!=, значение1, значение2)

Если числа *значение1* и *значение2* не равны, возвращается 1, в противном случае 0.

$(<=, значение1, значение2)

Если число *значение1* меньше или равно числу *значение2*, возвращается 1, в противном случае 0.

$(>=, значение1, значение2)

Если число *значение1* больше или равно числу *значение2*, возвращается 1, в противном случае 0.

$(and, значение1, значение2 [, значение3, ..., значение9])

Возвращается побитовое логическое произведение целых от *значение1* до *значение9*.

$(angtos, значение, [режим, точность])

Данное *значение* редактируется как угол в *режиме* и с *точностью*, определенными для аналогичной Лисп-функции. Если *режим* и *точность* опущены, текущие значения выбираются по заданным командой ЕДИНИЦЫ (UNITS).

$(edtime, время, представление)

Дата по юлианскому календарю, определенная *временем* (например, полученным с помощью функции $(getvar, date), редактируется в соответствии с *представлением*. Последнее состоит из строк форматов, которые заменяются представлением даты и времени. Символы, не интерпретируемые как строка формата, копируются буквально. Строки форматов приведены ниже. Допустим, что дата и время таковы:

Понедельник, 2 сентября 1994 4:53:17.506

Формат	*Вывод*	*Формат*	*Вывод*
D	2	H	4
DD	02	HH	04
DDD	Пон	MM	53
DDDD	Понедельник	SS	17
M	9	MSEC	506
MO	09	AM/PM	AM
MON	сен	am/pm	am
MONTH	сентября	A/P	A
YY	94	a/p	a
YYY	1994		

$(eq, значение1, значение2)

Если строки *значение1* и *значение2* совпадают, возвращается 1, в противном случае 0.

$(eval, строка)

Строка *строка* передается в вычислитель DIESEL-выражений, и возвращается результат вычисления.

$(fix, значение)

Вещественное *значение* усекается до целого путем удаления дробной части.

$(getenv, имя_переменной)

Возвращается значение переменной среды *имя_переменной*. Если *имя_переменной* не определено, возвращается пустая строка.

$(getvar, имя_переменной)

Возвращается значение системной переменной *имя_переменной*. Если *имя_переменной* не определено, возвращается пустая строка.

$(if, выражение, истина [, ложь])

Если *выражение* не равно нулю, вычисляется и возвращается выражение *истина*. В противном случае вычисляется и возвращается выражение *ложь*.

$(index, номер, строка)

Подразумевается, что аргумент *строка* содержит одно или несколько значений, разделенных запятой. Аргумент *номер* выбирает одно из этих значений для извлечений, начиная с первого элемента с номером 0. Эта функция может использоваться для извлечения значения X, Y или Z координат точки, полученной с помощью функции $(getvar), а также в приложениях для получения значений, хранящихся в системных переменных USERS1-5.

$(linelen)

Возвращает длину в символах самой длинной строки, которую может вывести данный монитор. С помощью этой функции можно изменять формат статусной строки в зависимости от возможностей монитора. Эта возможность используется только для конфигурации статусной строки с помощью переменной MODEMACRO.

$(nth, номер, аргумент0 [, аргумент1, ..., аргумент7])

Вычисляет и возвращает аргумент, выбранный по его номеру. Если номер равен 0, возвращается *аргумент0* и т. д. Следует обратить внимание на разницу между функциями $(nth) и $(index): $(nth) возвращает один из нескольких аргументов функции, в то время как $(index) извлекает одно из значений разделенной запятыми строки, переданной как один аргумент. Невыбранные значения не вычисляются.

$(or, значение1, значение2 [, значение3, ..., значение9])

Возвращается побитовая логическая сумма целых от *значение1* до *значение9*.

$(rtos, значение [, режим, точность])

Данное *значение* редактируется как вещественное число в формате, указанном *режимом* и *точностью*, определенными для аналогичных Лисп-функций. Если *режим* и *точность* опущены, текущие значения присваиваются с режимом и точностью, определенными командой ЕДИНИЦЫ (UNITS).

$(strlen, строка)

Возвращает длину *строки* в символах.

$(substr, строка, начало [, длина])

Возвращает подстроку из *строки*, начиная с символа *начало* на заданное *длиной* количество символов. Символы в строке нумеруются начиная с 1. Если *длина* опущена, возвращается вся оставшаяся часть строки.

$(upper, строка)

Возвращается строка, преобразованная в прописные буквы.

$(xor, значение1, значение2 [, значение3, ..., значение9])

Возвращается побитовое логическое исключающее ИЛИ целых от *значение1* до *значение9*.

Сообщения об ошибках

В зависимости от типа ошибки DIESEL-выражение вставляет указание об ошибке в поток вывода:

Сообщение	*Смысл*
$?	Синтаксическая ошибка (пропущена закрывающая скобка или незавершенная строка)
$(функ,??)	Неизвестный аргумент функции функ.
$(функ)??	Неизвестная функция функ.
$(++)	Избыточная длина строки вывода - выражение усечено

5.2. Переменная MODEMACRO

В 12-й версии за содержание статусной строки отвечает системная переменная MODEMACRO. Эта переменная позволяет определить и отредактировать данные для составления статусной строки в соответствии с точной спецификацией пользователя. При запуске Автокада эта переменная установлена в пустую строку. Она не сохраняется ни в рисунке, ни в файле конфигурации, ни где-либо еще. При необходимости задания переменной MODEMACRO определенного значения при каждом открытии рисунка можно загрузить требуемую установку переменной с помощью функции S::STARTUP, определенной в файле acad.lsp.

Переменная MODEMACRO определяется пользователем. Ее можно установить в любое строковое значение: по длине она ограничена возможностями Автолиспа, а по размеру - буфером связи Автолисп - Автокад. Команда УСТПЕРЕМ (SETVAR) дает возможность ее установки в диалоговом режиме с консоли, хотя при этом ее длина ограничивается 255 символами. При задании пустой строки (путем ввода символа точка (.) или ("") из Лисп-программы с помощью функции servar) Автокад отображает стандартную статусную строку.

Для отображения в статусной строке постоянной информации используется команда УСТПЕРЕМ (SETVAR) функция Автолиспа setvar. Для формирования изменяемой в процессе работы содержания статусной строки в нее следует включать "макровыражения", используя язык DIESEL.

Примеры статусной строки:

Команда:MODEMACRO

Новое значение MODEMACRO или . если нет <"">:
$(getvar, dwgname) Слой: $(getvar, clayer)

С помощью функции $(getvar, varname) можно получить значение любой системной переменной, текущее значение которой заменяет выражение в статусной строке. Теперь при переключении слоев переменная MODEMACRO переопределяется и, если она изменилась, на экран выводится новое имя слоя.

Выражения могут быть достаточно сложными и вложенными. Например, если требуется отобразить в статусной строке текущее значение шага привязки и ее угол в градусах, преобразование угла сетки шаговой привязки из радиан в градусы с отбрасыванием десятичных единиц можно осуществить с помощью вложенных выражений:

```
Команда:MODEMACRO
Новое значение MODEMACRO или . если нет <"">: Шаг: $(getvar, snapunit)
$(fix,$(*,$(getvar, snapang),$(/,180,3.14159)))
```

Еще лучше отобразить значения в текущих форматах для линейных и угловых единиц измерения:

```
Команда:MODEMACRO
Новое значение MODEMACRO или . если нет <"">: Шаг:
$(rtos,$(index,0,$(getvar,snapunit))),$(rtos,$(index,1,$(getvar,snapunit)))
$(angtos,$(getvar,snapang))
```

Введенные данные DIESEL копирует непосредственно на вывод, пока не встретится символ $ или строку в кавычках. Строки в кавычках можно использовать для подавления вычисления последовательности символов (без кавычек они рассматривались как DIESEL-функции). При использовании кавычек в тексте строки их вводят дважды ("").

Если текущий слой - "ФУНДАМЕНТ" и выполнена команда

```
Команда:MODEMACRO
Новое значение MODEMACRO или . если нет <"">: "$(getvar,dwgname)=""$(getvar,
        clayer)""
```

то в статусной строке будет написано

```
$(getvar,clayer)="ФУНДАМЕНТ"
```

Определение MODEMACRO в Автолиспе

Для формирования строки, передаваемой в системную переменную MODEMACRO, можно использовать Автолисп.

Пример:

```
(defun C:STATLINE ()
        (setvar "modemacro"
                (strcat "layer $(subsrtr,$(getvar,clayer),1,8)"
                "$(if,$(getvar,orthomode), ORTHO)"
                "$(if,$(getvar,snapmode), SNAP)"
                "$(if,$(getvar,tabmode), TEMPLET)"
                "$(if,$(=,tilemode), 0),$(if,$(=,$(getvar,svport),1),L))"
                )
        )
)
```

При выполнении этой программы будет создаваться статусная строка, аналогичная стандартной.

5.3. DIESEL-выражения

Строковые выражения на языке DIESEL можно применять в файлах меню и использовать как дополнительный способ создания макросов. Эти выражения могут возвращать строковые значения в ответ на стандартные команды Автокада, Лисп- и СРП-процедуры и другие макроопределения меню. Можно изменять вид и содержимое заголовков меню. При использовании в пунктах меню DIESEL-выражений необходимо соблюдать формат "$раздел=субменю", где $раздел - пункт меню, субменю - требуемое DIESEL-выражение. Часто применение макросов меню проще и яснее, чем использование Автолиспа, но не всегда.

Примеры использования DIESEL и Лиспа:

DIESEL-выражение

```
[Лист/Модель]^C^C^P$M=$(if,$(getvar,cvport),1),_mspace,_pspace)
```

Лисп-выражение

```
[Лист/Модель]^C^C^P(if(=(getvar "cvport")1)(command "_MSPACE")+
(command"_PSPACE"))(princ)
```

Оба пункта меню позволяют переключаться из пространства листа в пространство модели, но DIESEL-выражение короче и выполняется "прозрачно", не требуя вызова Лисп-функции (princ). Если ^P пропущена в обоих случаях, DIESEL-выражение будет отображать только введенную команду, в то время как Лисп-выражение будет отображать всю строку Автокада.

Если необходимо изменять заголовок в падающем меню, первым символом заголовка должен быть символ "$":

```
[$(eval,"Current  layer: "$(getvar,clayer))]
```

В этом случае заголовок пункта будет изменяться при изменении текущего слоя.

Еще один пример, в котором используется DIESEL-выражение одновременно для заголовка и части пункта меню, - практичный способ ввода текущих значений даты в рисунок:

```
[$(edtime,$(getvar,date),DDD", "D MON YYYY]^C^C_text \\\+
$M=$(edtime,$(getvar,date),DDD", "D MON YYYY);
```

Для отключения заголовка падающего меню ERASE (СОТРИ) во время действия команды можно использовать следующую строку:

```
[$(if, $(getvar,cmdactive),~)ERASE]_erase
```

Аналогично можно изменять пометку возле пункта падающего меню.

DIESEL-выражения в Автолиспе

В Лисп-процедурах DIESEL-выражения можно использовать с помощью вызова Лисп-функции menucmd. Формат аналогичен формату DIESEL-выражений в заголовках меню.

Пример фрагмента кода установки переменной c_time, равной текущему времени:

```
(setq c_time (menucmd "M=$(edtime,$(detvar,date),HH:MM a/p)"))
```

Для проверки возможностей языка DIESEL можно использовать, к примеру, вот такую программу на Автолиспе:

```
(defun C:DIESEL( /dsl)
        (while (/= dsl "M=")
                (setq dsl (strcat "M="(getstring T "\nDIESEL: ")))
                (princ(menucmd dsl))
        )
        (princ)
)
```

Отладка DIESEL-выражений - MACROTRACE

Данная системная переменная используется для отладки DIESEL-выражений. Обычно она отключена (установлена в нуль). При ее включении вычисление всех DIESEL-выражений отображается в зоне подсказок, включая вычислёние выражений в меню и статусной строке. Для устранения некоторых недоразумений в режиме отладки DIESEL-выражений переменную MODEMACRO установите в пустую строку, поскольку она будет тоже вычисляться и выводиться в зону подсказок.

Пакетные файлы и слайды

6.1. Пакеты команд

Эволюция Автокада от самых первых версий до последней, двенадцатой тесным образом коснулась принципов программированного составления чертежей. На первых этапах методика формирования графического файла была простой и незатейливой: составлялся текстовый файл с перечнем команд и опций Автокада, который последовательно выполнялся при загрузке или при вызове команды ПАКЕТ (SCRIPT). Такой пакетный файл создавался вне системы Автокада в ASCII-формате с помощью текстового редактора или текстового процессора. Этот файл имел расширение .scr. Данная методика получила весьма широкое распространение и, несмотря на широкие возможности Автолиспа и СРП-приложений в 12-й версии, в силу необходимости преемственности снизу вверх сохранилась до сих пор.

Итак, пакет команд позволяет считывать команды из текстового файла и выполнять заранее подготовленную последовательность команд Автокада. Пакет может быть вызван на исполнение с помощью специальной формы команды acad (при запуске) или с помощью команды ПАКЕТ (SCRIPT). С помощью этого средства можно создавать циклические демонстрации, составленные из сменяющихся изображений так называемых слайдов - моментальных фотографий графического экрана Автокада, полученных с помощью команды ДСЛАЙД (MSLIDE). В пакетные файлы можно включать комментарии для пояснения содержания и других замечаний. Любая строка, начинающаяся с точки с запятой (;), воспринимается как комментарий и игнорируется Автокадом при обработке пакетного файла.

В 12-й версии при вводе команд из пакета системные переменные PICKADD и PICKAUTO должны быть равны соответственно единице и нулю. Это позволяет обеспечивать совместимость с предыдущими версиями и упрощает адаптацию.

6.2. Вызов пакета при запуске Автокада

Для вызова пакета при запуске Автокада используется командная строка

```
дисковод>acad имя_рисунка пакетный_файл
```

Пакетный файл должен быть вторым файлом, указанным в командной строке вызова Автокада. При этом подразумевается, что он имеет расширение .scr. Если Автокад не может найти пакетный файл с указанным именем, на экране появляется соответствующее сообщение и для продолжения работы нужно нажать клавишу Enter.

Примером пакетного файла для 10-й версии Автокада может служить следующий файл:

```
1
home
Grid on
Ltscale 3.0
Layer s 0 color red 0
```

Если данную последовательность команд записать в файл с именем box.scr, то его можно вызвать следующим образом:

```
дисковод>acad x box
```

В пакетном файле задано имя рисунка ("home"), и имя рисунка по умолчанию "x", заданное в строке вызова "acad", будет проигнорировано. Пакет создает чертеж "home", устанавливает режим включенной сетки, задает масштаб типа линий 3.0, устанавливает слой "0" текущим и присваивает ему красный цвет. Затем Автокад переходит в режим рисования, приглашая вас к вводу команд.

В 12-й версии первые две строки пакетного файла не нужны. Если они присутствуют, Автокад 12-й версии воспринимает их как требование пакета вызвать соответствующий пункт главного меню и реагирует нужным образом.

Однако правильная работа пакетных файлов зависит от соответствующих последовательностей запросов команд Автокада 12. При работе с пакетами, разработанными для старых версий, следует учесть, что:

- пакеты, обращающиеся к пунктам главного меню 1(новый), 2(открой), 7(компиляция), 8(преобразование старого рисунка), 9(восстановление испорченного рисунка - для 11-й версии), совместимы с версией 12 и не требуют изменений;
- пакеты, обращающиеся к пункту главного меню 3(черчение), не соответствуют последовательности запросов команды ЧЕРТИ (PLOT) версии 12 и не будут работать правильно; то же относится и к запросам пункта 4(печать) главного меню;
- пакеты, обращающиеся к пункту главного меню 5(настройка), могут потребовать изменений для обеспечения соответствия последовательности запросов версии 12.

В 12-й версии имеется возможность использования пакета для свободного черчения или для изменения конфигурации Автокада. Для этого используются следующие форматы команд:

```
дисковод>acad -p имя_пакета
```

для свободного черчения;

```
дисковод>acad -r имя_чертежа имя_пакета
```

для изменения конфигурации.

6.3. Вызов пакета из Автокада

Команда ПАКЕТ (SCRIPT)

Для запуска пакета непосредственно из графического редактора используется команда ПАКЕТ (SCRIPT). В ответ на запрос Автокада следует ввести имя пакетного файла или нажать клавишу Enter для принятия имени текущего рисунка по умолчанию. Предполагается, что файл имеет стандартное расширение .scr; не следует включать его в ответ. Сначала выполняется последовательность команд из пакетного файла, а затем появляется подсказка Команда: (Command:). Если из пакетного файла считывается команда ПАКЕТ (SCRIPT), то работа с текущим пакетом прекращается и файл, заданный в данной команде, становится текущим пакетным файлом. Для ограничения запросов в пакете можно использовать системную переменную EXPERT. Поскольку она влияет на все выдаваемые Автокадом запросы, ее можно устанавливать прямо в пакете.

Команда ЗАДЕРЖИ (DELAY)

При использовании пакетов в демонстрационных целях иногда требуется задержать получаемое изображение для того, чтобы зритель мог успеть рассмотреть и осмыслить изображение на экране. Данная команда обеспечивает создание паузы между различными операциями Автокада.

Формат команды:

```
Команда: ЗАДЕРЖИ
Задержка в миллисекундах: число
```

Заданное число определяет длительность паузы. Чем больше число, тем больше пауза. Ввиду широкого диапазона скоростей обработки в компьютерных системах, довольно сложно дать точное определение времени задержки, однако шаг задержки должен составлять около 1 миллисекунды. Наибольшая задержка составляет 32767 (около полуминуты).

Команда ПРОДОЛЖИ (RESUME)

Если выполнение пакета было прервано, например, нажатием комбинации клавиш Ctrl+C или Backspace, то выполнение пакета с места прерывания можно продолжить этой командой.

Формат команды:

Команда: ПРОДОЛЖИ
Command: RESUME

Ошибка выполнения какой-либо команды пакета приведет к его прерыванию; если ошибка произойдет во время работы Автокада, то для продолжения также используется эта команда. Для продолжения выполнения пакета при ошибке внутри комады, выполняемой в пакете, команда ПРОДОЛЖИ вводится в "прозрачном" режиме, т. е. с апострофом.

Команды ГРАФЭКР (GRAPHSCR) и ТЕКСТЭКР (TEXTSCR)

Некоторые команды Автокада (например, СПИСОК и СТАТУС) автоматически переключают экран в текстовый режим в одномониторных режимах. Для возвращения в графический режим при выполнении пакета можно выполнить команду ГРАФЭКР (GRAPHSCR). Для перехода в текстовый режим используется команда ТЕКСТЭКР (TEXTSCR). В двухмониторных системах эти команды игнорируются.

Команда ВПАКЕТ (RSCRIPT) - циклические пакеты

В ряде случаев (при коммерческих демонстрациях, на выставках и т. д.) весьма удобны пакеты, завязанные в петлю, т. е. повторяющиеся неограниченное число раз. Для изготовления такого пакетного файла в него надо вставить в качестве последней команды команду ВПАКЕТ - возобнови пакет (RSCRIPT). В этом случае он будет вызываться повторно.

Пример:

РЕГЕН	*Регенерация рисунка*
ТЕКСТ 3,2 5.0 0	*Начальные установки для ввода текста*
АО "ДИАЛОГ-МИФИ" ВАС ПРИВЕТСТВУЕТ!	
Задержи 2000	*Задержка на 2 сек.*
СОТРИ П	*Стирание текста*
	Пробел для завершения выбора объектов
ВПАКЕТ	*Перезапуск пакета.*

Автолисп в пакетах

Пакетные файлы появились в ранних версиях Автокада, когда они были единственным средством программирования. Развитие Автокада привело к тому, что можно обходиться и без них. Однако и в 12-й версии, не говоря уже о 10-й и 11-й, есть смысл их использовать в различных приложениях. Возникает вопрос о возможности использования Автолиспа в пакетных файлах. Ответ здесь положительный - по сути, строка пакетного файла моделирует ввод какого-то текста в командную строку непосредственно. Если в строке файла поставить на первом месте открывающую скобку, то она интерпретируется как начало строки Автолиспа. Таким образом мы можем писать программы на Автолиспе и в пакетных файлах. Конечно, это более медленный способ ввода программ, но отдельные команды, например команды загрузки тех или иных Лисп-программ, можно очень эффектно использовать. Командные строки или BAT-файлы, содержащие запуск и выполнение EXE-модулей пользователя, позволяют вписать Автокад в более глобальную систему обработки информации, передавая промежуточную информацию через текстовые файлы. В 11-й и 12-й версиях сразу можно писать приложения на языке Си, но, как это часто бывает, большое количество ранее разработанных программ не хочется переделывать и выбрасывать и остается только один путь - использовать пакетные файлы в сочетании с программами пользователя и командными файлами для разработки комплексов обработки информации, в которые Автокад входит как одна из составляющих.

6.4. Использование слайдов

В Автокаде слайд представляет собой файл, содержащий "моментальный снимок" изображения на графическом мониторе. Для получения требуемого изображения можно использовать любые команды Автокада и затем сделать слайд. Слайды можно использовать в графических меню и диалоговых окнах, можно вызывать для просмотра на экране Автокада, компоновать презентационные диафильмы с помощью пакетного файла или с использованием проектора ANIMATOR.

Команда ДСЛАЙД (MSLIDE) - создание слайда

Эта команда позволяет создать слайд из примитивов, видимых на текущем видовом экране. Если в данный момент вы работаете в пространстве модели, создается слайд только текущего видового экрана.

Итак, сначала создается изображение, затем кадрируется на весь видовой экран, затем выполняется команда

Команда: ДСЛАЙД

Command: MSLIDE

Нужно задать имя слайда, после чего создается файл с указанным именем и расширением .sld. По ходу создания слайда производится "перерисовка" изображения, и за заданным процессом можно следить. В слайд не включаются те части рисунка, которые находятся вне экрана либо на отключенных или замороженных слоях.

При создании слайдов с тонированных изображений при большом размере видового экрана или на мониторах с высоким разрешением возможно появление черных линий по контурам объекта. Чтобы исключить побочный эффект при создании слайдов тонированных изображений, нужно использовать всю графическую зону экрана монитора с высоким разрешением.

Команда СЛАЙД (VSLIDE) - просмотр слайда

При необходимости просмотра слайда задается команда

Команда: СЛАЙД

Command: VSLIDE

Далее в 12-й версии выводится стандартное диалоговое окно файлов или в строке подсказок появляется запрос

Слайд-файл <по умолчанию>:

Имя текущего рисунка предполагается по умолчанию. Если файл содержится в библиотеке (имя_библиотеки.slb), используется формат

имя_библиотеки(имя_слайда)

и на текущем видовом экране появляется записанный ранее слайд. Восстановление текущего рисунка на мониторе обеспечивается командой ОСВЕЖИ (REDRAW).

Библиотеки слайдов

Из набора слайд-файлов можно создавать библиотеки слайдов, используя дополнительную сервисную программу SLIDELIB, расположенную в каталоге support пакета Автокад. Для создания библиотеки слайдов в командной строке операционной системы вводится команда

дисковод>slidelib имя_библиотеки [<список слайдов]

slidelib считывает список имен слайд-файлов из стандартного канала ввода. Обычно это делается переназначением стандартного канала ввода на файл, содержащий список слайд-файлов (по одному на строку

в файле, подготовленном с помощью какого-либо текстового редактора). Подготовленная библиотека записывается в файл с именем, заданным в поле имя_библиотеки с расширением .slb. Файлы, указанные в поле список слайдов, могут задаваться без расширения. Не следует удалять исходные файлы слайдов. Программа slidelib не позволяет обновлять библиотеку слайдов после ее создания. При необходимости добавить слайд в библиотеку или удалить из нее слайд следует изменить файл списка слайдов и повторить создание библиотеки. Размер библиотеки приблизительно равен сумме размеров содержащихся в ней файлов. Однако операционная система отводит под каждый файл на диске больше места, чем размер информации в файле, и, если размеры файлов небольшие, можно получить значительную экономию дисковой памяти.

Показ слайдов

Показать один слайд на текущем видовом экране можно при помощи команды СЛАЙД (VSLIDE). Если возникает необходимость показа демонстрации слайд-фильма, используется комбинация пакета с командой СЛАЙД. Скорость воспроизведения слайдов на мониторе обычно ограничена временем доступа к диску, которое требуется для считывания слайд-файла. Можно, однако, предварительно загрузить следующий слайд с диска в оперативную память за то время, пока рассматривается предыдущий файл, а затем быстро вывести этот слайд на экран из памяти. Для предварительной загрузки слайда необходимо перед именем слайда в команде СЛАЙД поставить звездочку. Следующая команда СЛАЙД учтет, что слайд был предварительно загружен, и выдаст его на экран не спрашивая имени файла.

Пример:

```
СЛАЙД SLIDE1
СЛАЙД *SLIDE2
ЗАДЕРЖИ 2000
СЛАЙД
СЛАЙД *SLIDE3
ЗАДЕРЖИ 2000
СЛАЙД
ЗАДЕРЖИ 3000
ВПАКЕТ
VSLIDE SLIDE1
VSLIDE *SLIDE2
DELAY 2000
VSLIDE
СЛАЙД *SLIDE3
DELAY 2000
VSLIDE
```

DELAY 3000
RSCRIPT

Последующий слайд загружается в память во время просмотра текущего слайда. Кроме того, используются дополнительные задержки по времени.

Как мы видим, слайды - это специфическое средство Автокада. При использовании слайдов следует руководствоваться следующими правилами:

- Слайды нельзя редактировать. Если во время показа слайда выполняются команды изменения рисунка (например, что-либо рисуется или редактируется), то они влияют не на слайд, а на текущий чертеж, который не виден. Поэтому во время показа слайда рекомендуется использовать только команды СЛАЙД (VSLIDE), ЗАДЕРЖИ (DELAY), ОСВЕЖИ (REDRAW), причем последняя команда возвращает текущий рисунок на экран.
- Для изменения слайда следует сначала отредактировать рисунок, с которого был сделан слайд, затем сделать новый слайд.
- Слои, цвета, увеличение масштаба изображения и другие параметры во время просмотра слайда не оказывают никакого воздействия на вид этого слайда.
- Если при создании слайда использовался графический монитор с низкой разрешающей способностью, а во время просмотра - монитор с высокой разрешающей способностью, то такой слайд все же можно просмотреть: Автокад соответствующим образом откорректирует образ. Однако этот слайд не будет использовать всех преимуществ до тех пор, пока он не будет переделан с исходного рисунка на мониторе с высоким разрешением.

Приложения для Автокада

Открытость системы Автокада широко используется для создания специализированных пакетов, используемых в различных приложениях, предназначенных для решения конкретных задач проектирования.

Широкой известностью пользуются программные продукты фирмы Softdesk для выполнения работ в архитектуре, гражданском и промышленном строительстве, картографии, машиностроении. Пакеты этой фирмы выполнены по модульному принципу, позволяющему каждому пользователю создавать свою среду проектирования.

7.1. Архитектура и строительство

Модули архитектурного проектирования представляют собой отдельные оверлейные структуры, которые позволяют выполнять определенные построения и решать заданный круг задач:

- Базовый модуль (Architectural Base)
  представляет собой набор инструментов для эскизного архитектурного проектирования. Он необходим для работы всех остальных приложений.
- Архитектурный модуль (Auto-Architect)
  является высокопроизводительным пакетом для профессионального архитектора, который содержит все необходимое для сквозного проектирования зданий практически любого типа. Инструментарий этого модуля позволяет формировать полный комплект документации на выполненный проект. Обеспечиваются дюймовые и метрические стандарты. Стандартное разнесение различных элементов по слоям поддерживается через диалоговые окна Автокада. Имеется значительное количество разных типов стен, потолочных перекрытий, дверей и окон, лестниц, кровель. Неограниченные возможности редактирования этих элементов позволяют создавать свои библиотеки.
- Инструментальный (Productivity Tools) модуль
  позволяет увеличить скорость разработки проекта за счет использования высокопроизводительного текстового редактора, электронных таблиц и калькулятора. Его можно использовать в среде Автокада отдельно или с другими модулями фирмы Softdesk.

- Модуль вентиляционного оборудования (HVAC)
представляет инструментарий для планировки отопления, вентиляции и систем кондиционирования воздуха. Создаваемые планы и диаграммы управления удовлетворяют самым современным требованиям стандартов по охране окружающей среды. Модуль работает совместно с архитектурным и инструментальным модулями.
- Модуль электрооборудования (Electric)
содержит набор символов, программных средств для разработки схем электроосвещения, сигнализации и связи, силовых и противопожарных. Модуль полностью интегрирован с архитектурным и инструментальным модулями.
- Модуль проектирования водопровода и канализации (Plumbing) позволяет разрабатывать планы водопровода, канализации и диаграмм управления. Модуль полностью интегрирован с архитектурным и инструментальным модулями. Содержит большое количество соответствующих символов.
- Модуль деталей (Details)
представляет собой набор средств для формирования спецификаций деталей и сборочных единиц. С помощью этого модуля можно формировать свои каталоги деталей, используемых в архитектуре, гражданском строительстве, машиностроении. Может использоваться как совместно с предыдущими модулями, так и только с Автокадом.

7.2. Машиностроение

Комплекс программных продуктов Genius 12 является удобным средством для проектирования изделий машиностроения в среде Автокада. В нем содержится большое количество дополнительных команд, непосредственно ориентированных на конструктора-разработчика, широкий выбор стандартных элементов конструкций и деталей (крепеж различного сортамента, опоры вращения, муфты, зубчатые колеса, фланцы и пр.). Управление процессом проектирования осуществляется через планшетное меню со сменными вкладышами. Каждый из модулей комплекса мажет быть адаптирован под нужды конкретного производства, парка станочного оборудования, действующие стандарты. Проектирование можно вести как на плоскости обычным образом, так и в объеме. Требуемые расчеты можно выполнять непосредственно в среде Автокада и формировать пояснительную записку либо в отдельном текстовом файле, либо в файле Автокада.

Послесловие

Ко времени выхода книжки в свет вышел Автокад 13-й версии. Его достоинства, особенности и привлекательность для пользователя еще требуют серьезного рассмотрения и исследования. Заранее не хочется делать какие-то выводы, однако, если вы хотите с ним познакомиться поближе с целью приобретения, свяжитесь с автором,и вы получите подробную информацию. 13-я версия работает в среде DOS, WINDOWS, WINDOWS NT. Ресурсы для работы 13 Автокада требуются более серьезные - 16 Мбайт для ДОС и 20 МБ для WINDOWS. В скором времени Автокад 13-й версии появится и на других платформах.

Основные отличия 13-й версии от предыдущих заключаются в следующем:

- команду PURGE (Удали) можно выполнить на любой стадии редактирования рисунка;
- при загрузке существующего чертежа в диалоговом окне можно просмотреть его вид;
- расчленить на составляющие можно любые блоки, вставленные с разными масштабами по осям;
- для Автокада в среде Windows имеется расширенная контекстная помощь;
- документация по всему пакету поставляется как в печатном виде изящно изданных более компактных книг, так и в электронном наборе гипертекстов, работающих в среде Windows;
- текстовые составляющие чертежа могут объединяться в параграфы, которые вставляются, редактируются и форматируются как единое целое;
- введена проверка орфографии для текста, к сожалению только англоязычного;
- можно использовать как в DOS-варианте, так и в Windows-варианте шрифты True-type в контурном и закрашенном виде;
- при простановке размеров можно вводить как величины отклонений размеров, так и использовать допуски отклонения формы, причем в разных стандартах;
- размерные стили получили дальнейшее усовершенствование - их можно вводить для разных типов размеров по группам;
- выноски стали ассоциативными и редактируются из команды DIM (РАЗМЕР);
- для каждой выноски можно составить аннотацию, редактируемую через диалоговое окно с подключением внешнего редактора;

- штриховка стала ассоциативной, и при изменении ее контуров она изменяется автоматически; введена новая команда редактирования штриховки, позволяющая изменять ее свойства через диалоговое окно;
- введены новые примитивы: эллипс, который заменил полилинейный квазиэллипс, сплайн, множественная линия, конструктивная линия, не имеющая ограничений по длине, луч - конструктивная линия, исходящая из точки в одну сторону;
- введены новые типы объектной привязки: пересечение продолжения, кажущееся пересечение примитивов, лежащих в разных плоскостях, и привязка через ссылочную точку;
- команды EXTEND, LENTHEN и TRIM работают не только с фактическими границами, но и с продолжениями примитивов;
- введены новые типы линий, содержащих символы или текстовые надписи;
- твердые тела стали примитивами Автокада, и к ним можно применять большинство команд редактирования;
- формат чертежа изменился, однако предусмотрена возможность экспорта в формате 12-й версии; экспортировать чертеж можно и в формате пакета 3D STUDIO, как, впрочем, и импортировать чертежи из этого пакета; чертежи предыдущих версий при загрузке конвертируются автоматически;
- в системе создания реалистичных изображений появились цветные источники света, источники света с ограниченным световым конусом;
- к каждому комплекту Автокада придается AutVision версии 2.0, который приближает процедуру и результат оттенения к работе пакета 3D-STUDIO.

Конечно, на этом отличия не заканчиваются. Возможности 13-й версии значительно шире. Это и новый драйвер дисплеев с широкой палитрой выбора цветовых гамм и разрешений, да еще и с дополнительными функциями зумирования типа "птичий глаз", новые драйверы принтеров-плоттеров и дигитайзеров. Вариант, работающий в среде Windows, позволяет широко пользоваться механизмами обмена и редактирования, специфическими для Windows. Кроме этого для этого варианта имеются свои новые возможности, например система частичных меню, подгружаемых по мере необходимости, встроенный курс обучения основам Автокада, документация в гипертекстовом режиме и пр. Мы не ставим своей целью полностью описать Автокад 13 (это будет, мы надеемся, следующая книга), однако уже сейчас можно сказать, что фирма Autodesk развивает свой самый известный в мире продукт весьма успешно.

Системные переменные Автокада

Ниже перечислены системные переменные Автокада. Системные переменные содержат информацию о настройках различных режимов Автокада и некоторую дополнительную информацию. Системные переменные можно считывать и изменять (если переменная не помечена "только для чтения") при помощи команды SETVAR (УСТПЕРЕМ) или из Автолиспа (getvar и setvar). В 12-й версии Автокада системную переменную можно вызвать, просто указав ее имя на подсказку "Команда". Системные переменные могут быть целого, вещественного, строкового типа или списком (тип каждой переменной указан в таблице). Большинство системных переменных Автокада записываются в файл чертежа. Если переменная записывается в файл конфигурации acad.cfg или вообще не сохраняется, это оговаривается особо.

ACADPREFICS

Строковая.

Содержит имя каталога, в котором Автокад будет искать вспомогательные файлы, если они не будут найдены в текущем каталоге (файлы прототипов, шрифтов, меню, программ Автолиспа и вставляемых рисунков). Если, например, вы работаете с проектом NUM_1, настройки которого хранятся в каталоге D:\ACAD\NUM_1, то перед загрузкой Автокада из каталога, скажем, WORK следует поместить в командный файл запуска Автокада строку

```
SET ACAD=D:\ACAD\NUM_1
```

и переменная ACADPREFIX получит значение

```
"D:\ACAD\NUM_1"
```

Только для чтения. Не сохраняется.

ACADVER

Строковая.

Содержит номер версии Автокада, который может принимать значение 10, 11, 12 или 10a, 11a, 12a. Эта переменная не идентична переменной SACADVER заголовка DXF-файла, в которой содержится номер уровня базы данных рисунка.

Только для чтения.

AFLAGS

Целая.

Биты этой переменной определяют флаги режимов создания атрибутов командой ATTDEF (АТОПР). Числа, приведенные ниже, устанавливают соответствующие биты этой переменной. Для установки нескольких битов следует записать сумму нужных чисел в AFLAGS:

1 Invisible (Скрытый),
2 Constant (Постоянный),
4 Verify (Контролируемый),
8 Preset (Установленный).

Не сохраняется.

ANGBASE

Вещественная.

Направление угла 0° в текущей ПСК.

ANGDIR

Целая.

Направление отсчета углов в текущей ПСК:

-1 по часовой стрелке,
0 против часовой стрелки.

APERTURE

Целая.

Высота прицела объектной привязки в пикселах. Записывается в файл конфигурации.

AREA

Вещественная.

Значение площади, вычисленной с помощью команд AREA (ПЛОЩАДЬ), LIST (СПИСОК) или DBLIST (БДСПИСОК).

Только для чтения. Не сохраняется.

ATTDIA

Целая.

Способ ввода значений атрибутов командой INSERT (ВСТАВЬ):

1 через диалоговое окно,
0 с командной строки.

ATTMODE

Целая.
Режим отображения атрибутов:
0 все атрибуты невидимы,
1 видимость определяется флагами,
2 видимы даже скрытые атрибуты.

ATTREQ

Целая.
Режим присвоения значений атрибутам при вставке командой INSERT (ВСТАВЬ):
0 ввод запрещен (берутся значения по умолчанию),
1 разрешается ввод значений в командной строке или через диалоговое окно.

AUDITCTL

Целая.
Управляет созданием файла отчета о проверке .adf:
0 файл отчета о проверке не создается,
1 включает запись ADT-файла отчета о проверке.

AUNITS

Целая.
Угловые единицы измерения:
0 десятичные градусы,
1 градусы/минуты/секунды,
2 грады,
3 радианы,
4 топографические единицы.

AUPREC

Целая.
Количество всех десятичных разрядов при отображении значений углов.

AXISMODE

Целая. 10-я и 11-я версии.
Отображение осей:

1 оси включены,
0 оси отключены.

AXISUNIT

Список из двух действительных чисел. 10-я и 11-я версии.

Цена деления в условных единицах разметки осей X и Y (команда AXIS (ОСИ)).

BACKZ

Вещественная.

Задает смещение задней секущей плоскости для текущего видового экрана в условных единицах. Определена только в том случае, если включен бит "заднего сечения" переменной VIEWMODE. Расстояние до задней секущей плоскости от камеры может быть найдено вычитанием переменной BACKZ из расстояния от камеры до цели.

Только для чтения.

BLIPMODE

Целая.

Отрисовка маркеров:

1 включена,
0 отключена.

CDATE

Вещественная.

Календарная дата/время (стандартный формат ОС UNIX).

Только для чтения.

CECOLOR

Строковая.

Текущий цвет.

Только для чтения.

CELTYPE

Строковая.

Текущий тип линии.

Только для чтения.

CHAMFERA

Вещественная.
Длина первой фаски.

CHAMFERB

Вещественная.
Длина второй фаски.

CIRCLERAD

Вещественная. Для 12-й версии.

Устанавливает радиус по умолчанию для кругов. При вводе 0 радиус по умолчанию не предполагается.

CLAYER

Строковая.
Текущий слой.
Только для чтения.

CMDDIA

Вещественная. Для 12-й версии.
В конфигурацию:
1 использовать в команде ЧЕРТИ (PLOT) диалоговые окна,
0 не использовать.

CMDACTIVE

Целая. Для 12-й версии.

Битовый код, определяющий, действует обычная и/или "прозрачная" команда, пакет или диалоговое окно. Является суммой следующего:

1 действует обычная команда,
2 действует обычная и "прозрачная" команда,
4 действует пакет,
8 действует диалоговое окно.

Только для чтения.

CMDECHO

Целая.

Отображение текста программы при выполнении функции Автолиспа:

1 не подавляется,
0 подавляется ("тихий" режим).

Не сохраняется.

CMDNAMES

Строковая. Для 12-й версии.

Отображает английское имя действующей команды (и "прозрачной" команды, если есть). Например,

LINE'ZOOM

COORDS

Целая.

Режим отображения координат в строке состояний (устанавливается клавишей "F6"):

0 отображаются только координаты точки указания,
1 отображаются текущие абсолютные координаты перекрестья,
2 при запросе угла или расстояния отображаются угол и расстояние от последней введенной точки.

CVPORT

Целая. Для 11-й версии.

Номер текущего видового экрана.

DATE

Вещественная.

Дата/время по Юлианскому календарю (стандартный формат ОС UNIX).

Только для чтения. Не сохраняется.

DIMxxx

Разные.

Размерные переменные. Все размерные переменные могут обрабатываться как системные переменные Автокада (см. гл. 3).

DBMODE

Целая. Для 12-й версии.

Битовый код, определяющий характер изменений в чертеже (только для чтения). Является суммой:

1 база данных примитивов чертежа изменена,
2 символьная таблица изменена,
4 переменные базы данных изменены,
8 рамка изменена,
16 вид изменен.

DIASTAT

Целая. Для 12-й версии.

Способ выхода из диалогового окна:

0 выход осуществлен с помощью клавиши "Отмена",
1 выход осуществлен с помощью клавиши "Да".

Только для чтения.

DISTANCE

Вещественная.

Расстояние, вычисленное командой DIST (ДИСТ).

Только для чтения. Не сохраняется.

DONUTID

Вещественная. Для 12-й версии.

Внутренний диаметр кольца по умолчанию. Может равняться нулю.

DONUTOD

Вещественная. Для 12-й версии.

Внешний диаметр кольца по умолчанию не может равняться нулю. Если DONUTID больше DONUTOD, их значения меняются местами при следующей команде.

DRAGMODE

Целая. В рисунок.

Режим слежения:

0 отключен,
1 включен (по запросу),
2 Авто.

DRAGP1

Целая. В конфигурацию.
Частота регенерации объекта при слежении.
Сохраняется в файле конфигурации.

DRAGP2

Целая. В конфигурацию.
Частота регенерации объекта при быстром слежении.
Сохраняется в файле конфигурации.

DWGCODEPAGE

Строковая. В рисунок. Для 12-й версии.

Кодовая таблица рисунка. При создании нового рисунка устанавливается равной системной кодовой таблице и далее не поддерживается. Должна отражать кодовую таблицу чертежа и может быть установлена в любое значение, допустимое для переменной SYSCODEPAGE, или значение "undefined". Сохраняется в заголовке.

DWGNAME

Строковая.

Имя рисунка. Может включать путь доступа операционной системы, если он был указан при вводе имени рисунка.

Только для чтения. Не сохраняется.

DWGPREFIX

Строковая.
Путь доступа операционной системы.
Только для чтения. Не сохраняется.

DWGTILED

Целая. Для 12-й версии.
Битовый код, определяющий, присвоено ли текущему чертежу имя:
0 у чертежа нет имени,
1 у чертежа есть имя.

Только для чтения

DWGWRITE

Целая. Для 12-й версии.

Управляет переключателем "только для чтения" в стандартном диалоговом окне файлов "Открытие рисунка" в команде ОТКРОЙ (OPEN):

0 открывает файл только для чтения,

1 открывает файл для чтения и записи (значение по умолчанию).

ELEVATION

Вещественная.
Текущий уровень.

ERRNO

Целая.
Код ошибки. Не сохраняется.

EXPERT

Целая.
Режим отображения подсказок типа "Вы уверены?":

0 все подсказки,

1 подавляются подсказки "Выполнять регенерацию?" и "Вы действительно хотите отключить текущий слой?",

2 подавляются перечисленные выше подсказки, а также "Блок уже существует. Переопределить его?" (команда BLOCK (БЛОК) и "Рисунок с этим именем уже существует. Заменить его?" (команда WBLOCK (ПБЛОК),

3 подавляются перечисленные выше подсказки, а также подсказки команды LTYPE (ТИПЛИН), если пользователь пробует загрузить уже загруженный тип линии или создать новый тип линии в файле, в котором этот тип линии уже определен,

4 подавляются перечисленные выше подсказки, а также подсказки команд UCS Save (ПСК Сохрани) и VPORTS Save (ВЭКРАН Запиши), если имя, вводимое пользователем, уже существует,

5 подавляются перечисленные выше подсказки, а также подсказки, выдаваемые "Размер Сохрани" и "Размер подави", если размерный стиль с указанным именем уже существует (для 11-й и 12-й версий). Подавление подсказки означает, что на нее автоматически дается положительный ответ.

Впоследствии можно будет использовать значения большие чем 5, тем самым расширяя круг подавляемых подсказок.

В зависимости от значения переменной EXPERT могут изменяться сценарии, макрокоманды меню, команды Автолиспа и функции команд.

Значение по умолчанию - 0. Не сохраняется.

EXTMAX

Список из двух действительных чисел.

Правая верхняя граница рисунка. Уменьшается только командами ZOOM All (ПОКАЖИ Все) или ZOOM Extents (ПОКАЖИ Границы).

Результат выдается в мировых координатах.

Только для чтения.

EXTMIN

Список из двух действительных чисел.

Левая нижняя граница рисунка. Уменьшается только командами ZOOM All (ПОКАЖИ Все) или ZOOM Extents (ПОКАЖИ Границы).

Результат выдается в мировых координатах.

Только для чтения.

FILEDIA

Целая.

1 использовать диалоговые окна, если это возможно,

2 не использовать диалоговые окна, за исключением случаев, когда с помощью тильды ("~") окно запрашивается в явном виде.

Записывается в файл конфигурации.

FILLETRAD

Вещественная.

Задает радиус сопряжения.

FILLMODE

Целая.

Управление режимом закрашивания:

1 включен,

0 отключен.

FLATLAND

Целая. Для 10-й версии.

Эта переменная служит для обеспечения совместимости трехмерных объектов, созданных старыми версиями Автокада, с данной версией; в последующем будет отменена:

1 объектная привязка, DXF-формат и Автолисп соответствуют старым версиям,

0 используются все новые возможности.

По умолчанию для новых рисунков принимается значение 0, а для старых - значение 1.

Для 11-й и 12-й версий отсутствует.

FRONTZ

Вещественная.

Определяет смещение передней секущей плоскости для текущего видового экрана в условных единицах. Расстояние до передней секущей плоскости от точки зрения определяется вычитанием переменной FRONTZ из расстояния от точки зрения до цели. Определена, только если одновременно включены бит "передняя секущая плоскость" переменной VIEWMODE и бит "передняя секущая плоскость не в точке зрения".

Только для чтения.

GRIDMODE

Целая.

Отображение сетки на текущем видовом экране:

1 включена,

0 отключена.

GRIDUNIT

Список из двух действительных чисел.

Задает интервалы сетки текущего видового экрана по X и Y.

GRIPBLOCK

Целая. Для 12-й версии.

В конфигурацию.

Управляет заданием ручек в блоках:

0 задается только одна ручка в точке вставки блока (значение по умолчанию),

1 ручки появляются у всех примитивов блока.

GRIPCOLOR

Целая 1-255. Для 12-й версии.
В конфигурацию.
Цвет невыбранных ручек в виде контура. Значение по умолчанию - 5 (синий).

GRIPHOT

Целая 1-255. Для 12-й версии.
В конфигурацию.
Цвет выбранных ручек в виде закрашенного квадрата. Значение по умолчанию - 1 (красный).

GRIPS

Целая. Для 12-й версии.
В конфигурацию.
Позволяет использовать ручки выбора примитивов для растягивания, перемещения, поворота, масштабирования и зеркального отражения:
0 ручки отключены,
1 ручки включены (задание по умолчанию).

Для настройки размера ручек и задания зоны притяжения графического курсора к ручке при объектной привязке используется переменная GRIPSIZE.

GRIPSIZE

Целая. Для 12-й версии.
В конфигурацию.
Размер квадрата, изображающего ручку.
Значение по умолчанию - 3.

HANDLES

Целая.
Режим присвоения меток примитивам:
0 отключен,
1 включен.

Только для чтения.

HIGHLIGHT

Целая.
Выделение объектов при выборе:
0 отключено,
1 включено.

Не сохраняется.

HPANG

Вещественная. Для 12-й версии.
Угол наклона образца штриховки по умолчанию.

HPDOUDLE

Целая. Для 12-й версии.
Штрихование крест-накрест для созданных пользователем штриховок:
0 обычная штриховка,
1 штриховка крест-накрест.

HPNAME

Строковая. Для 12-й версии.
Имя образца штриховки по умолчанию длиной до 34 символов без пробелов. Если значение по умолчанию не задано, возвращает "". Ввод точки (.) означает отсутствие значения по умолчанию.

HPSCALE

Вещественная. Для 12-й версии.
Масштабный коэффициент образца штриховки по умолчанию. Не может равняться нулю.

INSBASE

Список из двух действительных чисел.
Базовая точка вставки рисунка как блока в текущей ПСК (устанавливается также командой BASE (БАЗА)).

INSNAME

Строковая. Для 12-й версии.

Имя блока по умолчанию для команд DDINSERT (ДИАЛВСТАВЬ) и INSERT (ВСТАВЬ). Должно соответствовать соглашению об именах символов. Если значение по умолчанию не задано, возвращает "". Ввод точки (.) означает отсутствие значения по умолчанию.

LASTANGLE

Вещественная.

Конечный угол последней введенной дуги в плоскости XY текущей ПСК.

Только для чтения. Не сохраняется.

LASTPOINT

Список из двух действительных чисел.

Последняя введенная точка, координаты которой выражены в текущей ПСК (ссылка на нее выполняется знаком "@" во время ввода координат с клавиатуры).

Не сохраняется.

LASTPT3D

Список из трех действительных чисел.

Аналогична LASTPOINT. Эта переменная не используется в 11-й и 12-й версиях.

LENSLENGTH

Вещественная.

Задает фокусное расстояние (в мм), используемое при построении перспективы для текущего видового экрана.

Только для чтения.

LIMCHECK

Целая.

1 контроль лимитов включен,

2 отключен.

LIMMAX

Список из двух действительных чисел.

Задает строковые координаты правого верхнего угла лимитов.

LIMMIN

Список из двух действительных чисел.

Задает координаты левого нижнего угла лимитов, выраженные в мировых координатах.

LOGINNAME

Строковая. Для 12-й версии.

Отображает указанное при настройке или введенное при запуске Автокада имя пользователя.

Только для чтения.

LTSCALE

Вещественная.

Общий масштабный коэффициент типа линии.

LUNITS

Целая.

Определяет систему измерения линейных единиц:

1 научная,
2 десятичная,
3 техническая,
4 архитектурная,
5 с дробной частью.

LUPREC

Целая.

Задает число десятичных разрядов или знаменатель в линейных единицах.

MACROTRACE

Целая. Для 12-й версии.

Средство отладки для DIESEL-выражений:

0 отключает отладку (по умолчанию),
1 отображает вычисление DIESEL-выражений в зоне подсказок, включая выражения из меню и статусной строки.

MAXACTVP

Целая. Для 11-й и 12-й версий.

Максимальное число одновременно регенерируемых видовых экранов.

Только для чтения.

MAXSORT

Целая. Для 11-й и 12-й версий.

В конфигурацию.

Максимальное число символов или имен файлов, которое можно отсортировать с помощью команд вывода списков;

если общее число сортируемых элементов превышает это число, сортировка не производится (по умолчанию - 200). Записывается в файл конфигурации.

MENUCTL

Целая. Для 12-й версии.

В конфигурацию.

Управляет переключением "страниц" экранного меню при вводе команды с клавиатуры:

0 при вводе команды с клавиатуры "страницы" экранного меню не переключаются,

1 при вводе команды с клавиатуры "страницы" экранного меню переключаются.

MENUECHO

Целая.

Биты этой переменной управляют эхо-выводом и подсказками меню. Числа, приведенные ниже, устанавливают соответствующие биты этой переменной. Для установки нескольких бит следует записать в MENUECHO сумму нужных чисел:

1 подавляет эхо-вывод пунктов меню (аналогично переключателю ^P в макроопределениях меню),

2 подавляет системные подсказки во время действия меню,

4 подавляет переключатель ^P эхо-вывода пунктов меню.

По умолчанию принимается значение 0 (выдаются все пункты меню и системные подсказки).

Не сохраняется.

MENUNAME

Строковая.

Имя загруженного в данный момент файла меню. Включает путь доступа операционной системы, если он был указан.

Только для чтения.

MIRRTEXT

Целая.

Управление зеркальным отражением текста командой MIRROR (ЗЕРКАЛО):

0 ориентация текста не меняется,

не - 0 зеркальное отображение текста.

MENUMACRO

Строковая. Для 12-й версии.

Позволяет выводить в статусной строке текстовые строки, например имя текущего рисунка, время и дату или специальные режимы. Можно выводить как простые текстовые строки, так и специальные строки языка макросов DIESEL.

OFFSETDIST

Вещественная. Для 12-й версии.

Расстояние смещения по умолчанию.

При вводе отрицательного значения по умолчанию предполагается "Точка" ("Point").

OSMODE

Целая.

Биты этой переменной управляют режимами объектной привязки. Числа, приведенные ниже, устанавливают соответствующие биты этой переменной. Для установки нескольких бит следует записать в OSMODE сумму нужных чисел:

1	ENDpoint	(КОНточка),
2	MIDpoint	(СЕРедина),
4	CENter	(ЦЕНтр),
8	NODe	(УЗЕл),
16	QUAdrant	(КВАдрант),
32	INTersection	(ПЕРесечение),
64	INSert	(ТВСтавки),

128	PERpendicular	(НОРмаль),
256	TANgent	(КАСательная),
512	NEArest	(БЛИжайшая),
1024	QUICK	(БЫСтрый).

ORTHOMODE

Целая.
Управление режимом ORTHO (ОРТО):
1 включен,
0 отключен.

PDMODE

Целая.
Задает символ отображения примитива "точка" (см. гл. 3).

PDSIZE

Вещественная.
Размер символа отображения примитива "точка" (см. гл. 3).

PERIMETER

Вещественная.
Периметр, вычисленный командами AREA (ПЛОЩАДЬ), LIST (СПИСОК), DBLIST (БДСПИСОК).
Только для чтения. Не сохраняется.

PFACEMAX

Целая. Для 11-й версии.
Максимальное число вершин на одну грань.
Только для чтения.

PICKADD

Целая. Для 12-й версии.
В конфигурацию.
Управляет добавочным выбором примитивов:
0 отключает PICKADD. Последние выбранные индивидуально или с помощью рамки примитивы становятся новым набором выбора. Предварительно выбранные примитивы в набор не

добавляются. Добавить в набор примитивы можно, удерживая при выборе клавишу Shift в нажатом положении,

1 включает PICKADD. Все выбранные индивидуально или с помощью рамки примитивы добавляются в текущий набор выбора. Удалить примитивы из набора можно, удерживая при выборе клавишу Shift в нажатом положении.

Значение по умолчанию - 0.

PICKAUTO

Целая. Для 12-й версии.
В конфигурацию.

Управляет автоматическим созданием рамки выбора в ответ на запрос "Выберите объекты" ("Select objects"):

0 отключает PICKADD,
1 позволяет создавать рамку выбора (в том числе и секущую) автоматически на запрос "Выберите объекты" ("Select objects").

PICKBOX

Целая.
В конфигурацию.

Высота прицела выбора объекта в пикселах. Сохраняется в файле конфигурации.

PICKDRAG

Целая. Для 12-й версии.
В конфигурацию.

Управляет способом создания рамки выбора:

0 путем указания двух диагональных точек, определяющих рамку. При перемещении устройства указания из одной точки в другую его кнопки опущены (значение по умолчанию - 0),
1 нажав кнопку выбора устройства указания в первой из точек, переместиться к другой, удерживая кнопку выбора нажатой, и отпустить ее во второй.

PICKFIRST

Целая. Для 12-й версии.
В конфигурацию.

Управляет способом выбора примитивов, при котором можно сначала создать набор выбора, а затем ввести команду редактирования или справок:

0 отключает предварительный набор,

1 включает предварительный набор (по умолчанию - 1).

PLATFORM

Строковая. Для 12-й версии.

Сообщение только для чтения, указывающее используемую Автокадом среду.

PLINEGEN

Целая. Для 12-й версии.

В рисунок.

Управляет генерацией типа линии в вершинах двумерных полилиний. Если равна единице, штрихи не обрываются в вершинах полилинии. Если равна нулю, штрихи начинаются и обрываются в каждой вершине. Не влияет на сегменты полилиний, имеющих переменную ширину.

PLINEWID

Вещественная. Для 12-й версии.

В рисунок.

Ширина полилинии по умолчанию. Может равняться нулю.

PLOTID

Строковая. Для 12-й версии.

В конфигурацию.

Изменяет текущий настроенный плоттер на основе присвоенного ему описания. Делает текущим плоттер, описание которого начинается с заданного текста.

PLOTTER

Целая. Для 12-й версии.

В конфигурацию.

Изменяет плоттер по умолчанию на основе присвоенного ему целого (от 0 до максимально присвоенного). Можно создать до 29 конфигураций. Делает текущим плоттер с заданным номером конфигурации (номера от 1 до 28 присваиваются при настройке).

POLYSIDES

Целая. Для 12-й версии.

Количество сторон многоугольника по умолчанию для команды МН-УГОЛ (POLYGON). Допустимый диапазон от 3 до 1024.

POPUPS

Целая.

Признак наличия расширенного пользовательского интерфейса (возможность работы с диалоговыми окнами, строкой меню и падающими меню):

1 РПИ реализуется,

0 текущий драйвер монитора не поддерживает РПИ.

Только для чтения. Не сохраняется.

PSLTSCALE

Целая. Для 12-й версии.

В рисунок.

Управляет масштабом типа линий в пространстве листа:

0 специальное изменение масштаба типа линии отключено,

1 масштаб типа линий зависит от масштаба видового экрана.

PSPROLOG

Строковая. Для 12-й версии.

Задает имя пролога, считываемого из файла acad.pst при использовании команды ЭКСПОРТПС.

PSQUALITY

Целая. Для 12-й версии.

Управляет качеством отображения изображений PostScript и отрисовкой их контурными или закрашенными:

0 отключает генерацию изображений PostScript,

Положительное значение - задает количество пикселов на единицу измерения рисунка для разрешения PostScript,

Отрицательное значение - абсолютная величина задает количество пикселов на единицу рисунка. Изображения PostScript выводятся контурно (без закрашивания).

QTEXTMODE

Целая.
Режим отображения текста:
1 контурный текст,
0 нормальное отображение текста.

REGENMODE

Целая.
Автоматическая регенерация:
1 включена,
0 отключена.

RE-INT

Целая. Для 12-й версии.

Переинициализирует порты ввода/вывода, монитор, дигитайзер, плоттер и файл acad.pgp, используя описанные ниже битовые коды. Для задания нескольких переинициализаций следует ввести сумму значений кодов; например, 3 задает переинициализацию порта дигитайзера (1) и плоттера (2):

1 = порт дигитайзера,
2 = порт плоттера,
4 = дигитайзер,
8 = монитор,
16 = перезагрузка PGP-файла.

SAVEFILE

Строковая. Для 12-й версии.
Имя текущего файла для автоматического сохранения.
Только для чтения.

SAVENAME

Целая. Для 12-й версии.
Имя файла, в который копируется рисунок.
Только для чтения.

SAVETIME

Целая. Для 12-й версии.

Автоматическое сохранение текущего рисунка с расширением .sv$, с указанным интервалом в минутах (или 0 для отключения периодического сохранения). Таймер сбрасывается при вводе команд СОХРАНИ, СОХРАНИВ и БСОХРАНИ.

SCREENBOXES

Целая. Для 12-й версии.

Количество боксов в зоне экранного меню графического экрана. При отключенном экранном меню при настройке SCRINBOXES равна нулю. В системах, позволяющих изменять размер графического окна или перенастраивать экранное меню в течение сеанса редактирования, значение этой переменной может изменяться.

Только для чтения.

SCREENMODE

Целая. Для 12-й версии.

Битовый код, указывающий состояние экрана Автокада. Является суммой следующего:

0 = текстовый экран,
1 = графический экран,
2 = двухэкранная конфигурация.

Только для чтения.

SCREENSIZE

Список из двух действительных чисел.
Задает размер экрана графического монитора
в пикселах, X и Y.
Только для чтения. Не сохраняется.

SHADEDGE

Целая. Для 11-й и 12-й версий.

0 грани тонируются, ребра не выделяются,
1 грани тонируются, ребра рисуются фоновым цветом,
2 грани не заполняются, ребра рисуются цветом объекта,
3 грани рисуются цветом объекта, ребра - цветом фона.

SHADEDIF

Целая. Для 11-й и 12-й версий.

Отношение окружающей освещенности к освещенности диффузного отражения (в процентах от окружающей освещенности).

SHPNAME

Строковая. Для 12-й версии.

Имя формы по умолчанию. Должно соответствовать принятому соглашению об именах символов. Ввод точки (".") означает отсутствие значения по умолчанию.

SKETCHINC

Вещественная.

Шаг приращения в команде SKETCH (ЭСКИЗ).

SKPOLY

Целая.

Команда SKETCH (ЭСКИЗ) генерирует:

0 отрезки,

1 полилинии.

SNAPANG

Вещественная.

Угол поворота сетки шаговой привязки (в текущей ПСК) для текущего видового экрана.

SNAPBASE

Список двух действительных чисел.

Начальная точка сетки шаговой привязки для текущего видового экрана (в текущей ПСК).

SNAPISOPAIR

Целая.

Текущая плоскость изометрии текущего видового экрана:

0 левая,

1 верхняя,

2 правая.

SNAPMODE

Целая.
Режим SNAP (ШАГ) для текущего видового экрана:
1 включен,
0 отключен.

SNAPSTYL

Целая.
Задает тип текущего видового экрана:
0 стандартный,
1 изометрический.

SNAPUNIT

Список двух действительных чисел.
Задает шаг привязки по X и Y текущего видового экрана.

SORTENTS

Целая. Для 12-й версии.
Управляет порядком сортировки:
0 отключает переменную,
1 сортировка при выборе объектов,
2 сортировка при объектной привязке,
4 сортировка при перерисовках,
8 сортировка при создании слайдов,
16 сортировка при регенерациях,
32 сортировка при черчении,
64 сортировка при выводе в формате PostScript,
96 значение по умолчанию - сортировка при черчении и выводе в формате PostScript.

SPLFRAME

Целая.

Управление отображением каркаса сплайна при сглаживании полилинии, определяющей сети поверхности сглаживания (отображается либо поверхность сглаживания, либо сеть), а также невидимых краев 3M граней:
0 не отображаются,
1 отображаются.

SPLINESEGS

Целая.

Задает число прямых сегментов, которые должны быть сгенерированы для каждого участка сплайна.

SPLINETYPE

Целая.

Тип сплайна, создаваемого командой PEDIT Spline (ПОЛРЕД Сплайн):

5 квадратичный В-сплайн,
6 кубический В-сплайн.

Остальные значения запрещены.

SURFTAB1

Целая.

Задает число интервалов, которые должны быть сгенерированы в командах RULESURF (П-СОЕД) и TABSURF (П-СДВИГ), и плотность сети в направлении М для команд REVSURF (П-ВРАЩ) и EDGESURF (П-КРАЙ).

SURFTAB2

Целая.

Задает плотность сети в направлении N для команд REVSURF (П-ВРАЩ) и EDGESURF (П-КРАЙ).

SURFTYPE

Целая.

Тип поверхности сглаживания, создаваемой командой PEDIT Spline (ПОЛРЕД Сплайн):

5 поверхность квадратичного В-сплайна,
6 поверхность кубического В-сплайна,
8 поверхность Безье.

Остальные значения запрещены.

SURFU

Целая.

Плотность поверхности в направлении М.

SURFV

Целая.
Плотность поверхности в направлении N.

SYSCODEPAGE

Строковая. Для 12-й версии.
Указывает системную кодовую таблицу, заданную в файле acad.xmf.
Только для чтения.

TABMODE

Целая. Для 12-й версии.
0 отключает режим "планшет",
1 включает режим "планшет".

TARGET

Список из трех действительных чисел.
Задает положение цели в координатах ПСК на текущем видовом экране.
Только для чтения.

TDCREATE

Вещественная.
Время и дата создания рисунка (стандартный формат ОС UNIX).
Только для чтения.

TDINDWG

Вещественная.
Общее время редактирования (стандартный формат ОС UNIX).
Только для чтения.

TDUPDATE

Вещественная.
Время и дата последнего редактирования/записи (специальный формат, стандартный формат ОС UNIX).
Только для чтения.

TDUSRTIMER

Вещественная.
Таймер пользователя (стандартный формат ОС UNIX).
Только для чтения.

TEMPPREFIX

Строковая.

Путь операционной системы, по которому Автокад помещает временные файлы. Если у вас есть возможность использовать виртуальный (электронный) диск, то, указав в этой переменной путь доступа к нему, вы можете ускорить работу Автокада. Эта переменная не может быть установлена из графического редактора (следует воспользоваться подменю настройки главного меню).

Сохраняется в файле конфигурации.

TEXTEVAL

Целая.

Режим анализа ввода в тех запросах Автокада, где требуется ввод текстовых строк. Ввод выражений Автолиспа и передача значений переменных Автолиспа (знак "!"):

0 недопустим (будет воспринят как текст),
1 допустим (будут переданы Автолиспу).

Следует иметь в виду, что команда DTEXT (ДТЕКСТ) с Автолиспом не работает вообще.

Не сохраняется.

TEXTSIZE

Вещественная.

Задает высоту по умолчанию нового текста в текущей гарнитуре (если высота переменная).

TEXTSTYLE

Строковая.
Имя текущей гарнитуры шрифта.
Только для чтения.

THICKNESS

Вещественная.
Текущая высота примитивов.

TILEMODE

Целая. Для 11-й версии.

1 режим совместимости с версией 10 (использует команду ВЭКРАН),
0 активизирует пространство листа и режим работы с видовыми экранами, как с примитивами (использует команду СВИД).

TRACEWID

Вещественная.
Ширина полосы по умолчанию.

TREEDEPTH

Целая. Для 12-й версии.

Четырехзначный код, указывающий на количество делений на ветви древовидного пространственного индекса. Первые две цифры - глубина узлов в пространстве модели, две последние - глубина узлов в пространстве листа. Для трехмерных рисунков коды положительные, для двумерных - отрицательные.

TREEMAX

Целая. Для 12-й версии.

Ограничения максимального количества узлов в пространственном индексе рисунка (дерево октантов) для ограничения требования к памяти при регенерации.

UCSFOLLOW

Целая.

Режим слежения за изменением системы координат (автоматическое переключение на вид в плане в новой системе):

1 включен,
0 выключен.

UCSICON

Целая.

Биты этой переменной управляют отображением пиктограммы системы координат текущего видового экрана. Для установки бит следует записать в UCSICON нужное число:

1 пиктограмма отображается,
2 если отображается, то привязана к началу текущей ПСК (если это возможно).

UCSNAME

Строковая.
Содержит имя текущей системы координат.
Только для чтения.

UCSORG

Список из трех действительных чисел.
Начало текущей системы координат в мировых координатах.
Только для чтения.

UCSXDIR

Список из трех действительных чисел.
Задает направление X текущей ПСК.
Только для чтения.

UCSYDIR

Список из трех действительных чисел.
Задает направление Y текущей ПСК.
Только для чтения.

UNDOCTL

Целая. Для 12-й версии.
Код состояния команды ОТМЕНИ:
1 возможность отмены команд включена,
2 возможность отмены одной команды,
4 режим автогруппы включен,
8 действует группа.

Только для чтения.

UNDOMARKS

Целая. Для 12-й версии.
Количество активных меток, заданных командой ОТМЕНИ Метка.
Только для чтения.

UNITMODE

Целая.
0 отображение с дробной частью, в футах и дюймах и в топографических единицах, как ранее,
1 отображение с дробной частью, в футах и дюймах и в топографических единицах в формате ввода.

USERI1-5

Пять свободных целочисленных переменных для использования сторонними разработчиками.

USERR1-5

Пять свободных вещественных переменных для использования сторонними разработчиками.

USERS1-5

Строковая. Для 12-й версии.

Пять свободных переменных для запоминания строковых данных. Максимальная длина строки зависит от используемой системы, но не менее 460 символов.

VIEWCTR

Список из двух действительных чисел.
Центр текущего видового экрана в координатах ПСК.
Только для чтения.

VIEWDIR

Список из трех действительных чисел.

Направление взгляда на текущем видовом экране в мировых координатах. Описывает положение камеры вектором смещения от точки цели.

Только для чтения.

VIEWMODE

Целая.

Биты этой переменной управляют режимами отображения текущего видового экрана (1 - "Вкл."). Числа, приведенные ниже, в сумме определяют значение этой переменной:

1 перспективный вид,
2 переднее сечение,
4 заднее сечение,
8 переменная UCSFOLLOW равна единице,
16 переднее сечение не в точке зрения. Если этот бит установлен в единицу, то расстояние до переднего сечения (FRONTZ) определяет переднюю секущую плоскость. Если "Откл.", переменная FRONTZ игнорируется и передняя секущая плоскость проходит через точку установки камеры (т. е. векторы за камерой не отображаются). Этот флаг игнорируется, если бит "переднего сечения" (2) в состоянии "Откл."

Только для чтения.

VIEWSIZE

Вещественная.

Задает высоту изображения на текущем видовом экране в условных единицах.

Только для чтения.

VIEWTWIST

Вещественная.

Задает угол поворота вида для текущего видового экрана.

Только для чтения.

VISRETAIN

Целая. Для 12-й версии.

Управляет цветом и типом линий для слоев внешней ссылки:

0 первоначальное состояние (значение по умолчанию),
1 установка текущего рисунка.

VPOINTX,Y,Z

Вещественные. Для 10-й версии.

Задают направляющие векторы направления взгляда по осям X, Y и Z в текущем видовом экране в мировых координатах. Таким образом,

положение камеры описывается как вектор смещения от цели. Эти переменные не будут использоваться в следующих версиях, так как то же значение возвращается одной переменной VIEWDIR.

Только для чтения.

VSMAX

Список из двух действительных чисел.

Задает правый верхний угол "виртуального экрана" текущего видового экрана, выраженный в координатах ПСК.

Только для чтения.

VSMIN

Список из двух действительных чисел.

Задает левый нижний угол "виртуального экрана" текущего видового экрана, выраженный в координатах ПСК.

Только для чтения.

WORLDUCS

Целая.

Совпадение текущей ПСК с МСК:

1 совпадает,

0 не совпадает.

Только для чтения.

WORLDVIEW

Целая.

Система координат, в которой работают команды DVIEW (ДВИД) и VPOINT (ТЗРЕНИЯ), действующие в текущей ПСК:

1 текущая ПСК,

0 мировая система координат.

XREFCTL

Целая. Для 12-й версии.

Управляет записью файлов .xlg (файл журнала внешних ссылок):

0 файл не создается,

1 файл записывается.

Перечень команд Автокада

Таблица содержит список всех команд трех версий Автокада с кратким описанием их действий и опций. Команды для версий 11 и 12 помечены одним знаком "*", только для 12-й - "**".

АПЕРТУРА (APERTURE)

Устанавливает размер прицела режима объектной привязки.

АТОПР (ATTDEF)

Определяет атрибуты.

► Опции:

С(I)	Управляет видимостью атрибутов.
П(С)	Присваивает атрибуту статус "постоянный".
К(V)	Управляет режимом проверки.
У(Р)	Задает предварительные установки.

АТРЕД (ATTEDIT)

Средство редактирования атрибутов.

АТЭКР (ATTDISP)

Управляет видимостью атрибутов на экране.

► Опции:

В(ON)	Все атрибуты отображаются на экране.
О(OFF)	Все атрибуты невидимы.
Н(N)	Нормальный: видимость каждого атрибута определяется отдельно.

*АТЭКСП (ATTEXT)**

Экспортирует в файл данные, содержащиеся в атрибутах.

▶ Опции:

C	Экспорт в CDF-формате.
S	Экспорт в SDF-формате.
D	Экспорт в DXF-формате.
O(E)	Экспорт атрибутов из выбранных объектов.

БАЗА (BASE)

Задание базовой точки для последующей вставки блока в другой рисунок.

БЛОК (BLOCK)

Из группы объектов создает составной неделимый объект.

▶ Опции:

?	Отображает на экране список определенных в рисунке блоков.

БДСПИСОК (BDLIST)

Выводит на экран информацию, хранящуюся в базе данных, обо всех объектах рисунка.

*БСОХРАНИ (QSAVE) ***

Сохраняет рисунок без запроса имени файла.

ВЕРНИ (REDO)

Восстанавливает изменения, сделанные предыдущей командой, если это были команды О или ОТМЕНИ.

'ВИД ('VIEW)

Сохраняет текущее изображение на экране как именованный вид или изображает вид на экране.

▶ Опции:

У(D)	Удаляет именованный вид из списка сохраненных.
В(R)	Восстанавливает поименованный вид на экране.
С(S)	Сохраняет изображение на экране как поименованный вид.
Р(W)	Сохраняет как поименованный вид изображение в указанной рамке.
?	Список имен сохраненных видов.

ВОССТАН (RECOVER) **

Восстановление по возможности поврежденного или испорченного рисунка.

ВПАКЕТ (RSCRIPT)

Возобновляет исполнение прерванного пакета.

'ВРЕМЯ ('TIME)

Показывает время создания и редактирования рисунка и позволяет управлять таймером пользователя.

▶ Опции:

П(D)	Выводит на экран текущие значения таймера.
В(ON)	Включает таймер пользователя.
О(OFF)	Останавливает таймер пользователя.
С(R)	Сброс таймера пользователя.

'ВСЕОСВЕЖ ('REDRAWALL) *

Перерисовывает изображение на всех видовых экранах.

ВСЕРЕГЕН (REGENALL)

Регенерирует изображение на всех видовых экранах.

ВСЛОЙ (VPLAYER) *

Устанавливает видимость для новых и существующих слоев раздельно по видовым экранам.

▶ Опции:

?	Выводит список слоев замороженных на выбраном видовом экране.
З(F)	Замораживает указанные слои на выбранных видовых экранах.
Р(Т)	Размораживает указанные слои на выбранных видовых экранах.
С(R)	Изменяет видимость указанных слоев на значение по умолчанию.
Н(N)	Создает новый слой, замороженный на всех видовых экранах.
В(V)	Изменяет видимость по умолчанию на видовом экране для существующих слоев.

ВСТАВЬ (INSERT)

Вставляет копию блока в рисунок.

▶ Опции:

имя	Вставляет целый рисунок "имя-файла-рисунка" в текущий рисунок.
имя=f	Создает блок из целого рисунка "имя-файла-рисунка".
*имя	Сохраняет графические примитивы отдельных частей.
?	Выводит на экран список определенных в рисунке блоков (в ответ на запрос масштаба по оси X).
Y	Определяет масштаб указанием двух точек (угловое задание масштаба) (в ответ на запрос масштаба по оси X).
XYZ	Воспринимается как необходимость задания масштабов по осям X, Y, Z.

ВЫБЕРИ (SELECT)

Составляет из выбранных объектов набор для последующего выполнения команд.

ВЭКРАН (VPORTS)

Разделяет графический экран на несколько частей, каждая из которых может содержать собственный вид рисунка.

▶ Опции:

У(D)	Удаляет из списка сохраненных указанную конфигурацию видовых экранов.
С(J)	Соединяет (сливает) два видовых экрана в один.
В(R)	Восстанавливает сохраненную конфигурацию видовых экранов.
З(S)	Записывает имя текущей конфигурации экрана в список сохраненных.
О(SI)	Отключает видимость всех видовых экранов, кроме одного текущего.
2	Делит текущий видовой экран на два видовых экрана.
3	Делит текущий видовой экран на три видовых экрана.
4	Делит текущий видовой экран на четыре видовых экрана.
?	Список имен текущей и сохраненных конфигураций видовых экранов.

'ГРАФЭКР ('GRAPHSCR)

При одноэкранной конфигурации переключает монитор в графический режим; команда употребляется в пакетах и при создании меню.

ДАКОМ (REDEFINE)

Восстанавливает встроенную команду, переопределенную с помощью команды НЕТКОМ.

ДВИД (DVIEW)

Определяет параллельные и перспективные проекции видов в динамическом режиме.

▶ Опции:

К(CA)	Выбор угла поворота камеры относительно цели.
СЕ(CL)	Устанавливает заднюю и переднюю секущие плоскости.
Р(D)	Устанавливает расстояние от камеры до цели, включает перспективу.
СК(H)	Убирает скрытые линии в наборе.
ОТК(OFF)	Отключает перспективное изображение.
ПА(PA)	Панорамирование рисунка.
Т(PO)	Задает точки расположения камеры и цели.
Ц(TA)	Вращает вид вокруг направления взгляда.
В(TW)	Вращает вид вокруг направления взгляда.

ОТМ(U)	Отменяет действие субкоманды ДВИД.
X(X)	Прерывает выполнение команды ДВИД.
ПО(Z)	Зуммирование или задание фокусного расстояния.

*ДЕРЕВО (TREESTAT) ***

Отображает информацию о текущем пространственном индексе рисунка, такую, как количество и глубина узлов в базе данных. Используется вместе с системной переменной TREEDEPTH для увеличения производительности при работе с большими рисунками.

*ДИАЛАТОП (DDATTDEF) ***

Выводит на экран диалоговое окно для создания определения атрибута, включаемого в качестве текстовой информации в определение блока (см. АТОПР).

'ДИАЛАТР ('DDLMODES)

Редактирование атрибутов через диалоговое окно.

*ДИАЛАТЭК (DDATTEXT) ***

Выводит на экран диалоговое окно для извлечения данных из рисунка. Допустимыми форматами являются DXF, CDF и SDF (см. АТЭКСП).

*ДИАЛВСТ (DDINSERT) ***

Выводит на экран диалоговое окно, позволяющее вставить копию предварительно созданного стандартного элемента или файла в текущий рисунок (см. ВСТАВЬ).

*'ДИАЛВЫБ ('DDSELECT) ***

Вызывает диалоговое окно для задания режимов выбора примитивов, величину прицела выбора и способ сортировки объектов.

*'ДИАЛЕДИН ('DDUNITS) ***

Вызывает диалоговое окно для задания форматов отображения и точности координат и углов (см. ЕДИНИЦЫ).

*ДИАЛИМЯ (DDRENAME) ***

Вызывает диалоговое окно для переименования гарнитуры шрифтов, слоев, типов линий, блоков, видов, ПСК, конфигураций видовых экранов и размерных стилей (см. НОВОЕ ИМЯ).

*'ДИАЛПРИВ ('DDOSNAP) ***

Вызывает диалоговое окно для установления текущих режимов объектной привязки и изменения величины прицела выбора объектов для привязки (см. ПРИВЯЖИ).

'ДИАЛПРИМ ('DDEMODES)

Установка текущего слоя, цвета, типа линии и высоты через диалоговое окно.

ДИАЛПСК (DDUCS)

Изображает на экране диалоговое окно для управления ПСК.

*'ДИАЛРАЗМ ('DDIM) ***

Отображает на экране серию диалоговых окон для управления простановкой размеров (см. РАЗМЕР).

*ДИАЛРЕД (DDEDIT) **

Выводит на экран диалоговое окно для редактирования текстов и определений атрибутов.

*'ДИАЛРУЧ ('DDGRIPS) ***

Позволяет через диалоговое окно включать ручки выбора примитивов и задавать их цвет и величину.

*ДИАЛСВОЙ (DDCHPROP) ***

Позволяет через диалоговое окно изменять цвет, слой, тип линии и высоту выбранных примитивов (см. СВОЙСТВА).

'ДИАЛСЛОЙ ('DDLMODES)

Установка характеристик слоя через диалоговое окно.

'ДИАЛСРЕД ('DDRMODES)

Установка режимов рисования через диалоговое окно.

ДИСТ (DIST)

Определяет расстояние между двумя точками.

ДОБАВЬ (XBIND) *

Добавляет выбранный набор зависимых от внешней ссылки символов в текущий рисунок.

► Опции:

Б(B)	Блок.
Р(D)	Размерный стиль.
С(LA)	Слой.
Т(LT)	Тип линии.
Г(S)	Гарнитура.

ДОС (SHELL)

Предоставляет возможность запуска других программ без выхода из Автокада.

ДОС1 (SH)

Предоставляет доступ к внутренним командам ДОС без выхода из Автокада.

ДСЛАЙД (MSLIDE)

Записывает изображение на экране в слайд-файл.

ДТЕКСТ (DTEXT)

Рисует на экране текстовые символы заданного размера и начертания.

► Опции:

В(J)	Запрашивает опции выключки.
ВПИ(F)	Текст заданной гарнитуры, вписанный между двумя точками; высота определяется автоматически.
Ц(C)	Горизонтальное центрирование строки текста.

ВЫ(A)	Строка текста заданной высоты выравнивается между двумя точками; соответствующая степень растяжения/сжатия вычисляется автоматически.
С(M)	Горизонтальное и вертикальное центрирование строки текста.
П(R)	Строка текста выравнивается по правому краю.
Г(S)	Выбор гарнитуры шрифта.
НЛ(BL)	Выключка вниз влево.
НЦ(BC)	Выключка вниз в центр.
НП(BR)	Выключка вниз вправо.
СЛ(ML)	Выключка посередине влево.
СЦ(MC)	Выключка посередине в центр.
СП(MR)	Выключка посередине вправо.
ВЛ(TL)	Выключка вверх влево.
ВЦ(TC)	Выключка вверх в центр.
ВП(TR)	Выключка вверх вправо.

ДУГА (ARC)

Рисует дугу любого радиуса; метод задания дуги, принимаемый по умолчанию, заключается в указании на две ее конечные точки и на точку, принадлежащую дуге.

► Опции:

У(A)	Центральный угол.
Ц(C)	Центр.
Н(D)	Направление.
К(E)	Конец.
Д(L)	Длина хорды.
Р(R)	Радиус.
RETURN	Воспринимается (при нажатии в ответ на запрос "Центр") как указание совместить начало дуги с концом последней нарисованной дуги или отрезка.

'ЕДИНИЦЫ ('UNITS)

Выбор формата и точности представления координат и углов.

ЗАГРУЗИ (LOAD)

Загружает файл, содержащий описание определенных пользователем форм.

▶ Опции:

?	Выдает на экран список загруженных файлов описания форм.

'ЗАДЕРЖИ ('DELAY)

Приостанавливает выполнение команды на указанное время; употребляется в пакетных файлах.

'ЗАКРАСЬ ('FILL)

Определяет, будут ли закрашены изображаемые на экране или выводимые на бумагу полосы, фигуры и полилинии.

▶ Опции:

В(ON)	Полосы, фигуры и полилинии закрашены.
О(OFF)	Полосы, фигуры и полилинии только оконтурены.

ЗЕРКАЛО (MIRROR)

Рисует зеркальное отображение указанного объекта.

ЗНАКПСК (UCSICON)

Управляет видимостью пиктограммы пользовательской системы координат.

▶ Опции:

В(ON)	Включение отображения пиктограммы ПСК.
О(OFF)	Отключение отображения пиктограммы.
ВС(А)	Внесение изменений на все видовые экраны.
Б(N)	Помещать пиктограмму в левый нижний угол видового экрана.
Н(OR)	Размещает пиктограмму ПСК по возможности в начальной точке текущей ПСК.

ИЗМЕНИ (CHANGE)

Изменяет положение, размер, ориентацию в пространстве и другие характеристики выбранного объекта. Особенно интересна при работе с текстом.

▶ Опции:

С(Р)	Общие свойства примитивов.

СЛ(LA)	Слой.
Ц(C)	Цвет.
Т(LT)	Тип линии.
В(T)	Высота.
У(E)	Уровень.

'ИЗОМЕТР ('ISOPLANE)

Задает изометрическую плоскость для рисования.

► Опции:

Л(L)	Левая плоскость.
П(R)	Правая плоскость.
В(T)	Верхняя плоскость.
RETURN	Делает следующую по очереди плоскость текущей.

ИМПОРТА (DXFIN)

Загружает файл обмена рисунками.

ИМПОРТД (DXBIN)

Вставляет закодированный в двоичном формате файл в рисунок.

ИМПОРТИ (IGESIN)

Загружает обменный файл в формате IGES.

*ИМПОРТПС (PSIN) ***

Загружает файл в формате Encapsulated PostScript (EPS).

*'ИНФО ('ABOUT) ***

Через диалоговое окно выводит различную информацию, в том числе о версии Автокада, серийном номере и текстом из файла acad.msg.

КОЛЬЦО (DONUT)

Рисует кольца с заданным внешним и внутренним диаметром.

*КОМПИЛ (COMPILE) ***

Компилирует файлы форм и шрифтов.

КОНЕЦ (END)

Выход из рисунка с сохранением внесенных изменений.

'КООРД ('ID)

Выдает на экране в зоне подсказки координаты указанной точки.

КОПИРУЙ (COPY)

Рисует копии указанного объекта.

▶ Опции:

Н(М)	Создает несколько копий указанного объекта.

*КПОЛИ (BPOLY) ***

Создает полилинию по замкнутому контуру.

КРУГ (CIRCLE)

Рисует круг любого заданного радиуса; по умолчанию принят метод задания круга по центру и точке на его окружности.

▶ Опции:

2Т(2Р)	Задает круг по двум конечным точкам на его диаметре.
3Т(3Р)	Задает круг по трем точкам на его окружности.
Р(R)	Задает круг по радиусу.
Д(D)	Для указания диаметра вместо запрашиваемого радиуса.
ККР(TTR)	Задание круга указанием двух касательных и радиуса.

КТЕКСТ (QTEXT)

При включенном режиме на экране появляются не символы, а контуры текста.

▶ Опции:

В(ON)	Режим контурного текста включен.
О(OFF)	Режим контурного текста отключен.

*КШТРИХ (BHATCH) ***

Автоматически заполняет определенный контур образцом штриховки с использованием диалогового окна, а также помогает предварительно просматривать и многократно подгонять штриховку без выхода из команды.

ЛИМИТЫ (LIMITS)

Изменяет лимиты рисунка и проверяет соблюдение этих лимитов.

► Опции:

2 точки	Устанавливает лимиты по нижнему левому и верхнему правому углам рисунка.
В(ON)	Включает проверку лимитов.
О(OFF)	Отключает проверку лимитов.

*ЛИСТ (PSPACE) **

Делает текущим пространство листа.

ЛМАСШТАБ (LTSCALE)

Устанавливает масштаб для всех типов линий, появляющихся в рисунке.

МАРКЕР (BLIPMODE)

Управляет видимостью маркеров на экране при указании точек.

► Опции:

В(ON)	На экране появляются временные маркеры.
О(OFF)	Маркеры не появляются.

МАССИВ (ARRAY)

Производит многократное копирование выбранного объекта, располагая копии прямоугольным или круговым массивом.

► Опции:

К(Р)	Круговой массив.
П(R)	Прямоугольный массив.

МАСШТАБ (SCALE)

Пропорционально изменяет размеры существующих объектов.

► Опции:

C(R)	Изменение размера со ссылкой на величину существующего объекта.

МВСТАВЬ (MINSERT)

Вставляет массив копий блока в рисунок.

► Опции:

имя	Вставляет целый рисунок "имя-файла-рисунка" и формирует из вставленных рисунков массив.
имя=f	Создает блок из целого рисунка "имя-файла-рисунка" и формирует из созданных блоков массив.
?	Выводит на экран список определенных в рисунке блоков (как ответ на запрос масштаба по оси X).
У	Определяет масштаб указанием двух точек (угловое задание масштаба) (как ответ на запрос масштаба по оси X).
XYZ	Воспринимается как необходимость задания масштабов по осям X, Y, Z.

МЕНЮ (MENU)

Загружает в рисунок файл меню, содержащий команды Автокада (экранное меню).

МЕТКИ (HANDLES)

Присваивает уникальный постоянный номер каждому примитиву рисунка.

► Опции:

В(ON)	Присваивает метки всем примитивам и устанавливает переменную HANDLES, равную единице.
У(D)	Удаляет ранее присвоенные примитивам метки.

МНОГОРАЗ (MULTIPLE)

Вызывает повторение последующей за этой спецификацией команды до тех пор, пока она не будет прервана.

МН-УГОЛ (POLIGON)

Рисует правильные многоугольники с заданным числом сторон.

► Опции:

С(Е)	Задание многоугольника указанием одной его стороны.
О(С)	Описанный вокруг окружности.
В(I)	Вписанный в окружность.

*МОДЕЛЬ (MSPACE) **

Делает текущим пространство модели.

НАСТРВИД (VIEWRES)

Заданием числа сторон изображающего на экране круг многоугольника определяется точность изображения и скорость регенерации кругов и дуг.

*НАСТРОЙ (CONFIG) ***

Позволяет изменять на текстовом экране настройки монитора дигитайзера, плоттера и рабочих параметров.

НЕТКОМ (UNDEFINE)

Переопределяет встроенную команду.

НОВОЕИМЯ (RENAME)

Изменяет имена гарнитур шрифтов, слоев, типов линий, блоков и видов ПСК и конфигураций видовых экранов.

► Опции:

Б(В)	Изменяет имя блока.
С(LA)	Изменяет имя слоя.
Т(LT)	Изменяет имя типа линии.
Г(S)	Изменяет имя гарнитуры.
П(U)	Изменяет имя ПСК.
ВИ(VI)	Изменяет имя вида.
ВЭ(VP)	Изменяет имя конфигурации видовых экранов.
Р(D)	Изменяет имя размерного стиля.

*НОВЫЙ (NEW) ***

Создает новый рисунок.

О (U)

Отменяет действие предыдущей команды.

ОБРЕЖЬ (TRIM)

Удаляет части примитивов, которые пересекают указанную границу.

ОЙ (OOPS)

Восстанавливает стертые ранее примитивы.

'ОРТО ('ORTHO)

Включает режим рисования примитивов только параллельно сетке.

'ОСВЕЖИ ('REDRAW)

Перерисовывает текущий видовой экран.

ОСИ (AXIS)

Только для Автокада 10.

Отображает на экране монитора две шкалы графической зоны.

▶ Опции:

О(OFF)	Отключает видимость направляющих линий.
В(ON)	Включает видимость направляющих линий.
Ш(S)	Устанавливает интервал делений, равный шагу привязки.
А(А)	Устанавливает цену деления по оси X, отличную от цены деления по оси Y.
число	Устанавливает цену деления (0=устанавливается равной шагу привязки).
числоX	Устанавливает цену деления кратно шагу привязки.

ОТМЕНИ (UNDO)

Отменяет действие нескольких команд; имеет более широкие возможности, чем команда О.

▶ Опции:

число	Отменяет действие указанного количества команд начиная с последней.
А(А)	Авто: любая операция из меню интерпретируется командами О и ОТМЕНИ как один шаг.
О(В)	Обратно: восстанавливает состояние рисунка на момент проставления контрольной точки.
У(С)	Управление: определяет количество шагов, запоминаемых командами О и ОТМЕНИ.
К(Е)	Конец: последовательность команд начиная с метки.
Г(G)	"Группа" и заканчивающаяся меткой "КОНЕЦ" воспринимается командами О и ОТМЕНИ как одна команда.
М(М)	Метка: устанавливает метку в протоколе рисунка.
В(А)	Включает все средства ОТМЕНИ.
Н(N)	Полностью отключает команды О и ОТМЕНИ и отнимает всю информацию команды ОТМЕНИ, записанную ранее в сеансе редактирования.
О(О)	Ограничивает действие команды О или ОТМЕНИ одной операцией.

ОТРЕЗОК (LINE)

Рисует прямолинейный отрезок указанной длины.

► Опции:

RETURN	(Как ответ на запрос "От точки".) Начало изображаемого отрезка совмещается с концом последней нарисованной дуги или отрезка (как ответ на запрос "К точке:").
З(C)	Замыкает ломаную (как ответ на запрос "К точке:").
O(U)	Отменяет последний нарисованный сегмент.

П-ВРАЩ (REVSURF)

Создает трехмерную многоугольную сеть аппроксимацией поверхности вращения, полученной в результате вращения кривой вокруг заданной оси.

П-КРАЙ (EDGESURF)

Создает трехмерную многоугольную сеть методом аппроксимации участка поверхности Кунса (бикубическая поверхность, полученная интерполяцией между четырьмя краями).

П-СДВИГ (TABSURF)

Создает многоугольную сеть, аппроксимирующую поверхность, полученную сдвигом определяющей кривой вдоль направляющего вектора.

П-СОЕД (RULESURF)

Создает трехмерную многоугольную сеть аппроксимацией поверхности между двумя кривыми.

ПАКЕТ (SCRIPT)

Выполняет указанный командный пакет.

'ПАН ('PAN)

Перемещает окно экрана по рисунку.

ПБЛОК (WBLOCK)

Записывает изображение указанных объектов в отдельный файл на диск.

▶ Опции:

имя	Запись определенного ранее блока в отдельный файл.
=	Присвоение файлу имени выписываемого блока.
*	Запись на диск всего рисунка.
RETURN	Запись указанных объектов.

ПГРАНЬ (PFAGE) **

Создает трехмерную сеть произвольной сложности с произвольными характеристиками поверхности.

ПЕРЕИН (REINIT) **

Повторно инициализирует порты ввода/вывода, дигитайзер, монитор, плоттер и PGP-файл.

ПЕРЕНЕСИ (MOVE)

Параллельный перенос выбранного объекта.

ПЕЧАТАЙ (PRPLOT)

Только для Автокада 10.

Выводит рисунок на матричный принтер. В более поздних версиях эти функции выполняет команда ЧЕРТИ.

ПЛАН (PLAN)

Устанавливается точка зрения (0, 0, 1).

▶ Опции:

Т(С)	План текущей ПСК.
П(U)	План указанной ПСК.
M(W)	План МСК.

ПЛАНШЕТ (TABLET)

Настраивает планшет для точного копирования изображения на бумаге средствами Автокада.

► Опции:

В(ON)	Включает режим "Планшет".
О(OFF)	Отключает режим "Планшет".
К(CAL)	Калибровка планшета.
Н(CFG)	Настройка зон планшетного меню.

ПЛИНИЯ (PLINE)

Рисует двумерные полилинии (последовательности из прямолинейных и дуговых сегментов с возможным указанием их ширины).

► Опции:

П(Н)	Устанавливает новую полуширину.
ОТМ(U)	Отменяет последний нарисованный сегмент.
Ш(W)	Устанавливает ширину.
RETURN	Выход из команды ПЛИНИЯ.
	В режиме отрезка:
ДУ(А)	Переключение в режим дуги.
З(С)	Замыкание прямолинейным сегментом.
ДЛ(L)	Длина сегмента (продолжает предыдущий сегмент).
	В режиме дуги:
У(А)	Центральный угол.
Ц(СЕ)	Центральная точка.
З(CL)	Замыкание прямолинейным сегментом.
Н(D)	Направление.
ОТР(L)	Длина хорды или переключение в режим отрезка.
Р(R)	Радиус.
В(S)	Вторая точка дуги по трем точкам.

ПЛОЩАДЬ (AREA)

Вычисляет площадь многоугольника, замкнутой полилинии, круга.

► Опции:

Д(А)	Устанавливает режим добавления.
В(S)	Устанавливает режим вычитания.
П(Е)	Вычисляет площадь выбранного объекта.

ПОВЕРНИ (ROTATE)

Поворачивает существующий объект.

▶ Опции:

С(R)	Поворот относительно существующего угла.

ПОДЕЛИ (DIVIDE)

Делит объект на заданное число частей, помечая их маркерами.

▶ Опции:

Б	Вместо маркера вставляется указанный блок.

ПОДОБИЕ (OFFSET)

Создает подобные существующим кривые и параллельные линии.

▶ Опции:

число	Определение расстояния смещения.
Т(Т)	Точка, через которую будет проходить линия, подобная заданной.

'ПОКАЖИ ('ZOOM)

Уменьшает или увеличивает изображение объекта на экране.

▶ Опции:

число	Масштаб по отношению к первоначальному изображению объекта на экране.
числоХ	Масштаб по отношению к текущему изображению объекта на экране.
числоХЛ (ХР)	Масштаб относительно пространства листа.
В(А)	Все.
Ц(С)	Центр.
Д(D)	Динамика.
Г(Е)	Границы.
Л(L)	Нижний левый угол.
П(Р)	Предыдущий.
Р(W)	Рамка.
М(V)	Максимальное уменьшение без регенерации.

ПОКИНЬ (QUIT)

Выход из графического редактора и возвращение в Главное меню без записи сделанных за сеанс изменений.

ПОЛОСА (TRACE)

Рисует закрашенные линии заданной ширины.

ПОЛРЕД (PEDIT(2D))

Редактирует двумерные полилинии.

► Опции:

З(С)	Замыкает открытую полилинию.
У(D)	Убирает сглаживание, т. е. возвращает полилинию после применения сглаживания в первоначальное состояние.
В(Е)	Редактирует вершины.
СГ(F)	Сглаживает полилинию, делая из ломаной кривую.
Д(J)	Добавляет к существующей полилинии.
Т(L)	Включает генерацию линии с обрывом или без обрыва штрихов в вершинах.
РА(О)	Разрывает в указанной вершине замкнутую полилинию.
СП(S)	Использует вершины полилинии как фрейм для сплайн-аппроксимации (тип определяется значением переменной SPLINETYPE).
О(U)	Отменяет последнее действие редактирования.
Ш(W)	Устанавливает ширину всех сегментов.
Х(Х)	Выход из ПОЛРЕД.
	В режиме редактирования вершин:
РА(В)	Устанавливает первую вершину для разрыва.
В(G)	Выполни (разрыв или выпрямление).
ВС(J)	Восставляет новую вершину после текущей.
ПЕ(М)	Переносит текущую вершину в указанную точку.
С(N)	Устанавливает следующую по порядку вершину текущей.
П(Р)	Устанавливает предыдущую вершину текущей.
РЕ(R)	Регенерирует изображение полилиний.
ВЫ(S)	Устанавливает первую вершину для выпрямления.
К(Т)	Устанавливает касательное направление для текущей вершины.
Ш(W)	Устанавливает новую ширину для следующего сегмента.
Х(Х)	Выход из команды или прерывание процесса выпрямления или сглаживания.

ПОЛРЕД(3М) (PEDIT(3D))

Редактирование трехмерных полилиний.

▶ Опции:

З(С)	Замыкает открытую полилинию.
У(D)	Убирает сглаживание.
В(Е)	Редактирование вершин (субкоманды описаны ниже).
РА(О)	Разрывает в указанной вершине замкнутую полилинию.
СП(S)	Использует вершины полилинии как каркас для сплайн-аппроксимации (тип определяется значением переменной SPLINETYPE).
О(U)	Отменяет последнее действие редактирования.
Х(Х)	Выход из команды ПОЛРЕД.
	В режиме редактирования вершин:
РА(В)	Устанавливает первую вершину для разрыва.
В(G)	Выполни (разрыв или выпрямление).
ВС(J)	Вставляет новую вершину после текущей.
ПЕ(М)	Переносит текущую вершину в указанную точку.
С(N)	Устанавливает следующую по порядку вершину текущей.
П(Р)	Делает текущей предыдущую вершину.
РЕ(R)	Регенерирует полилинию.
ВЫ(S)	Устанавливает первую вершину для выпрямления.
Х(Х)	Выход из процесса редактирования или прерывание выпрямления или сглаживания.

ПОЛРЕД (сеть) (PEDIT) (Mesh)

Редактирует трехмерные многоугольные сети.

▶ Опции:

У(D)	Убирает сглаживание и восстанавливает первоначальный вид сети.
В(Е)	Редактирует вершины полилинии.
М	Размыкает или замыкает сеть в направлении М.
Н(N)	Размыкает или замыкает сеть в направлении N.
С(S)	Сглаживает поверхность (тип определяется значением SURFTYPE).
О(U)	Отменяет последнюю операцию редактирования.
Х(Х)	Выход из команды ПОЛРЕД.

'ПОМОЩЬ ('HELP)

Выводит на экран список команд и форматы ввода данных; можно применять для получения справки по указанной команде.

'ПРИВЯЖИ ('OSNAP)

Геометрически точное совмещение указываемых точек с характерными точками объектов.

► Опции:

ЦЕН(CEN)	К центру дуги или круга.
КОН(END)	К ближайшей конечной точке отрезка или дуги.
ТВС(INS)	К точке вставки текста, блока, формы.
ПЕР(INT)	К точке пересечения отрезков, дуг, окружностей.
СЕР(MID)	К середине дуги, отрезка.
БЛИ(NEA)	К ближайшей видимой точке дуги, круга, отрезка или точке.
УЗЕ(NOD)	К узлу (точке).
НИЧ(NON)	Ничего (Откл.).
НОР(PER)	Нормаль к дуге, отрезку или кругу.
КВА(QUA)	К квадранту дуги или круга.
БЫС(QUI)	Быстрый режим (возвращается не ближайшая к перекрестью, а первая найденная точка).
КАС(TAN)	Касательная к дуге или окружности.

ПРОВЕРЬ (AUDIT) *

Выполняет проверку целостности рисунка.

► Опции:

Д(Y)	Исправляет замеченные ошибки.
Н(N)	Только сообщает о замеченных ошибках.

'ПРОДОЛЖИ ('RESUM)

Продолжает прерванный процесс выполнения пакета.

ПЗАКР (PSFILL) **

Заполняет контуры (двумерные полилинии) образцом закраски формата PostScript, определенным в файле поддержки PostScript Автокада (akad.pst).

ПСК (UCS)

Определяет или видоизменяет текущую ПСК.

► Опции:

У(D)	Удаляет одну или более записанных систем.
O(E)	Устанавливает ПСК с тем же направлением выдавливания, что и у указанного объекта.
H(O)	Перемещает точку отсчета текущей ПСК.
П(P)	Делает предыдущую ПСК текущей.
З(R)	Заменяет текущую ПСК на записанную ранее.
C(S)	Сохраняет текущую ПСК.
B(V)	Поворачивает текущую ПСК, совмещая направление взгляда с направлением оси Z.
M(W)	Устанавливает текущую ПСК, эквивалентную МСК.
X	Поворачивает текущую ПСК вокруг ее оси X.
Y	Поворачивает текущую ПСК вокруг ее оси Y.
Z	Поворачивает текущую ПСК вокруг ее оси Z.
ZO(ZA)	Задание ПСК по точке отсчета и положительному направлению оси Z.
3	Задание ПСК по трем точкам: началу отсчета, точке на положительном направлении оси X и аналогичной точке на оси Y.
?	Список имен сохраненных ПСК.

ПССЛЕДИ (PSDRAG) **

Управляет режимом слежения при динамическом размещении изображения, импортируемого из PostScript-файла командой ИМПОРТПС.

► Опции:

К(0)	При слежении отображается только габаритный прямоугольник.
1	При слежении отображается тонированное изображение.

РАЗМЕР (DIM)

Выход в режим проставления размеров, позволяющий дополнить рисунок размерными и выносными линиями.

РАЗМЕР1 (DIM1)

Вход в режим проставления одного размера, затем выход в обычный командный режим.

РАЗМЕТЬ (MEASURE)

Ставит метки на выбранном объекте через указанные интервалы.

► Опции:

Б(B)	Вместо метки будет вставлен указанный блок.

РАЗОРВИ (BREAK)

Стирает часть объекта или разрывает его на две части.

► Опции:

П(F)	Переопределяет первую указанную точку.

РАСТЯНИ (STRETH)

Позволяет растягивать объект, перемещая одну из его частей, но не разрывая его.

РАСЧЛЕНИ (EXPLODE)

Расчленяет блок или полилинию на составляющие элементы, не изменяя геометрии объекта, но удаляя определение блока или полилинии.

РЕГЕН (REGEN)

Регенерирует изображение в текущем видовом экране.

'РЕГЕНАВТО ('REGENAUTO)

Определяет, будет ли произведена регенерация, обусловленная выполнением другой команды.

▶ Опции:

В(ON)	Автоматическая регенерация без сообщений.
О(OFF)	Регенерация с разрешения пользователя.

СВИД (MVIEW)

Создает видовые экраны и управляет ими.

▶ Опции:

В(ON)	Включает выбранные видовые экраны и регенерирует на них изображения объекта.
О(OFF)	Отключает изображение на выбранных видовых экранах.
С(H)	Удаляет невидимые линии при выводе на плоттер в пространстве листа.
ВП(F)	Создает единый видовой экран, вписанный в текущий вид пространства листа.
2	Создает два видовых экрана, вписанных в заданную область.
3	Создает три видовых экрана, вписанных в заданную область.
4	Создает четыре равных видовых экрана, вписанных в заданную область.
П(R)	Переводит конфигурации видовых экранов, сохраненные командой ВЭКРАН, в совокупности примитивов пространства листа.
точка	Создает новый видовой экран внутри области, заданной двумя точками.

СВОЙСТВА (CHPROP)

Изменяет свойства выбранных объектов.

▶ Опции:

Ц(C)	Цвет.
Т(LT)	Тип линии.
СЛ(LA)	Слой.
В(T)	Высота.

'СЕТКА ('GRID)

Изображает на экране сетку из точек с заданным расстоянием между ними.

▶ Опции:

В(ON)	Включает видимость сетки.
О(OFF)	Отключает видимость сетки.
Ш(S)	Устанавливает интервал равным шагу привязки.
А(А)	Устанавливает отличные друг от друга интервалы по осям.
число	Устанавливает интервал сетки (0=шаг привязки).
числоХ	Задает интервал сетки кратным шагу привязки.

СКРОЙ (HIDE)

Удаляет скрытые линии на трехмерном изображении объекта.

СЛАЙД (VSLIDE)

Изображает на экране содержимое ранее созданного слайд-файла.

▶ Опции:

имя	Показывает на экране содержимое файла "имя".
*имя	Загружает слайд-файл для употребления следующей командой СЛАЙД.

'СЛЕДИ ('DRAGMODE)

Включает и отключает режим динамического отслеживания, применяющийся при выполнении некоторых команд.

В(ON)	Ввод требований "СЛЕДИ" при необходимости.
О(OFF)	Игнорирование требования "СЛЕДИ".
А(А)	Устанавливает автоматический режим: отслеживание включается без запроса.

'СЛОЙ ('LAYER)

Создает в текущем рисунке именованные слои и присваивает им цвет и тип линии.

▶ Опции:

Ц(C)	Присваивает указанному слою цвет.
З(F)	Замораживает слои.
Т(L)	Присваивает указанному слою тип линии.
С(M)	Делает текущим слой; если такого слоя не существует, создает его.
Н(N)	Создает новые слои.
В(ON)	Включает видимость слоев.
О(OFF)	Отключает видимость слоев.
У(S)	Устанавливает текущим существующий слой.
Р(T)	Размораживает слои.
?	Выдает на экран список определенных в рисунке слоев, их состояние, цвет и тип линии.
А(U)	Разблокирует слой.

СОПРЯГИ (FILLET)

Построение сопряжения двух отрезков, дуг или кругов заданного радиуса.

▶ Опции:

ПОЛ(P)	Сопряжение между собой отдельных сегментов одной полилинии.
РАД(R)	Установка радиуса сопряжения.

СОТРИ (ERASE)

Удаляет примитивы из рисунка.

СОХРАНИ (SAVE)

Записывает сделанные в сеансе редактирования изменения в файл, не выходя из графического редактора.

СОХРАНИВ (SAVEAS) **

Аналогично команде СОХРАНИ, но при этом переименовывает текущий рисунок.

СПИСОК (LIST)

Выводит на экран информацию о выбранном объекте, содержащуюся в базе данных рисунка.

ССЫЛКА (XREF) *

Использует другие рисунки Автокада без их фактического добавления в текущий рисунок и без изменения их содержания.

► Опции:

В(А)	Вставляет новую внешнюю ссылку или копию вставленной ранее либо освежает изображение обновленного вставленного рисунка.
Д(В)	Делает внешнюю ссылку постоянной частью текущего рисунка.
У(D)	Удаляет внешнюю ссылку из текущего рисунка.
П(Р)	Позволяет просмотреть и отредактировать имя файла, используемое при вставке внешней ссылки.
O(R)	Обновляет одну или несколько внешних ссылок без перегрузки рисунка и освежает их изображение.
?	Перечисляет все внешние ссылки в текущем рисунке и рисунок, связанный с каждой из них.

'СТАТУС ('STATUS)

Выводит на экран текущую информацию о рисунке и режимах.

'СТИЛЬ ('STYLE)

Создание новых гарнитур шрифта, пользователю предоставляется возможность выбрать шрифт, его направление, наклон и степень растяжения.

► Опции:

?	Выводит на экран список гарнитур, определенных в рисунке.

ТЕКСТ (TEXT)

Позволяет создавать в рисунке текстовую информацию, которая выводится на экран после завершения команды.

Подробное описание опций - см. команду ДТЕКСТ.

'ТЕКСТЭКР ('TEXTSCR)

При одноэкранной конфигурации переключает в текстовый режим монитор; команда употребляется в пакетных файлах и при создании меню.

ТЕНЬ (SHADE) *

Строит тонированное изображение модели в текущем видовом экране.

ТЗРЕНИЯ (VPOINT)

Выбор в пространстве точки зрения на объект.

► Опции:

П(R)	Определение точки зрения заданием двух углов поворота.
RETURN	Выбор точки зрения поворотом тройки осей координат.
x,y,z	Задание точки зрения ее координатами.

'ТИПЛИН ('LINETYPE)

Определение типа линии как последовательности штрихов, точек и пробелов; загрузка его из библиотеки и присвоение последующим нарисованным объектам.

► Опции:

?	Вывод на экран списка имен типов линий, содержащихся в библиотеке.
С(С)	Создание определения типа линии.
З(L)	Загрузка определения типа линии.
У(S)	Установка типа линии текущим для последующего рисования.

ТОЧКА (POINT)

Рисует отдельные точки.

УДАЛИ (PURGE)

Удаляет из рисунка ненужные блоки, формы, гарнитуры шрифтов, слои, блоки и типы линий.

► Опции:

В(A)	Удаляет все ненужные поименованные объекты.
Б(B)	Удаляет ненужные блоки.
С(LA)	Удаляет ненужные слои.
Т(LT)	Удаляет ненужные типы линий.
Ф(SH)	Удаляет ненужные формы.
Г(ST)	Удаляет ненужные атрибуты.
Р(D)	Удаляет ненужные размерные стили.

УДЛИНИ (EXTEND)

Удлиняет отрезок, дугу или полилинию до пересечения с другим объектом.

УРОВЕНЬ (ELEV)

Устанавливает уровень и высоту для вновь создаваемых объектов.

'УСТПЕРЕМ ('SETVAR)

Дает возможность вывести на экран или изменить значения системных переменных.

'ФАЙЛЫ ('FILES)

Осуществляет работу с файлами на диске.

ФАСКА (CHAMFER)

Снимает фаску с угла, образованного пересечением двух отрезков.

► Опции:

Д(L)	Устанавливает длину фасок.
ПОЛ(P)	Снимает фаску с полилинии.

ФИГУРА (SOLID)

Рисует закрашиваемые многоугольники.

ФИЛЬМ (FILMROOL)

Создает файл для дальнейшей его загрузки в АВТОШЕЙД.

ФОРМА (SHAPE)

Рисует ранее определенные формы.

► Опции:

?	Список имен доступных форм.

'ЦВЕТ ('COLOUR)

Устанавливает цвет вновь создаваемого примитива.

► Опции:

число	Устанавливает цвет объекта.
имя	Устанавливает цвет объекта; стандартное имя цвета.
ПОБ(BYB)	Присваивает цвет "ПОБлоку".
ПОС(BYL)	Присваивает цвет "ПОСлою".

ЧЕРТИ (PLOT)

Вывод рисунка на плоттер.

'ШАГ ('SNAP)

Устанавливает шаг для ввода точек с помощью дигитайзера, что облегчает указание и делает его точным.

► Опции:

число	Устанавливает шаг привязки.
B(ON)	Включает режим ШАГ.
O(OFF)	Отключает режим ШАГ.
A(A)	Устанавливает различный шаг по разным осям.
П(R)	Поворачивает сетку шаговой привязки.
T(S)	Выбор типа изометрического стиля.

ШТРИХ (HATCH)

Осуществляет штриховку и заполнение областей.

► Опции:

имя	Использует для штриховки образец, имя которого указано.
C(U)	Использует для штриховки простой образец, создаваемый пользователем.
?	Выводит на экран названия имеющихся образцов.

ЭКСПОРТА (DXFOUT)

Записывает файл обмена рисунками на диск.

► Опции:

Д(V)	Записывает файл в двоичном формате.
О(E)	Только выбранные объекты.
0-16	Количество знаков после запятой.

ЭКСПОРТИ (IGESOUT)

Записывает на диск обменный файл в формате IGES.

*ЭКСПОРТПС (PSOUT) ***

Экспортирует текущий вид рисунка в файл в формате Encapsulated PostScript (EPS)

ЭЛЛИПС (ELLIPSE)

Рисует эллипсы.

► Опции:

Ц(C)	По центру.
П(R)	По эксцентриситету указанием угла поворота вокруг второй оси.
И(I)	Рисует круг на текущей изометрической плоскости.

ЭСКИЗ (SKETCH)

Выполнение эскизов "от руки".

► Опции:

Р(C)	Продолжает эскизную линию начиная с конечной точки предыдущей линии.
С(E)	Стирает временные (незаписанные) эскизные линии.
П(P)	Поднимает/опускает перо.
О(Q)	Выход из команды с отменой временных линий.
З(R)	Запись временных линий без выхода из команды.
Х(X)	Запись временных линий с последующим выходом из команды.
	Рисует линию до указанной устройством указания точки.

3-ГРАНЬ (3DFACE)

Рисует участок трехмерной плоскости.

▶ Опции:

Н(I)	Делает указанный край невидимым.

3-ПОЛИ (3DPOLY)

Создание трехмерных полилиний.

▶ Опции:

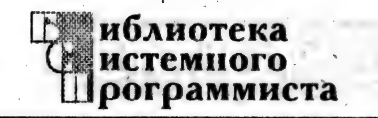
З(С)	Замыкает открытые полилинии.
О(U)	Отменяет (удаляет) последний нарисованный сегмент.
RETURN	Выход из команды.

3-СЕТЬ (3DMESH)

Задает трехмерную многоугольную сеть указанием ее размера (в терминах M, N) и положение каждой ее вершины.

Содержание

Из серии:
А. В. Фролов,
Г. В. Фролов



Том 11. ОПЕРАЦИОННАЯ СИСТЕМА MICROSOFT WINDOWS 3.1.
Введение для программиста. Часть 1

Том 12. ОПЕРАЦИОННАЯ СИСТЕМА MICROSOFT WINDOWS 3.1.
Для программиста. Часть 2

Том 13. ОПЕРАЦИОННАЯ СИСТЕМА MICROSOFT WINDOWS 3.1.
Для программиста. Часть 3

Том 14. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС GDI В MICROSOFT WINDOWS.

Том 15. МУЛЬТИМЕДИА ДЛЯ WINDOWS.
Руководство программиста

Том 16. МОДЕМЫ И ФАКС-МОДЕМЫ.
Программирование для MS-DOS и Windows

Том 17. MICROSOFT WINDOWS 3.1 Для программиста.
Дополнительные главы

Том 18. MS-DOS ДЛЯ ПРОГРАММИСТА.Часть 1

Том 19. MS-DOS ДЛЯ ПРОГРАММИСТА.Часть 2

Том 20. ОПЕРАЦИОННАЯ СИСТЕМА IBM OS/2 WARP

Том 21. Программирование видеоадаптеров EGA,VGA и SVGA

Том 22. ОПЕРАЦИОННА СИСТЕМА WINDOWS 95. Для программиста



Серия книг для самостоятельно постигающих тайны компьютерного мира. Авторы
А. В. Фролов и Г. В. Фролов

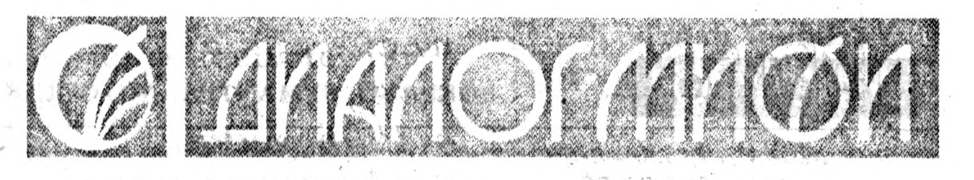
Том 1. ВВЕДЕНИЕ В MS-DOS, MS Windows, MS Word for Windows.

Том 2. ОПЕРАЦИОННАЯ СИСТЕМА Microsoft Windows.
Руководство пользователя

Том 3. СЕТИ КОМПЬЮТЕРОВ В ВАШЕМ ОФИСЕ

Том 4. ЧТО ВЫ ДОЛЖНЫ ЗНАТЬ О СВОЕМ КОМПЬЮТЕРЕ

Том 5. ОСТОРОЖНО: КОМПЬЮТЕРНЫЕ ВИРУСЫ!


ДИАЛОГ МИФИ